Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Aleš Papler

Vtičnik za pomnjenje nastavitev sprejemanja piškotkov na spletu

diplomsko delo

univerzitetni študijski program prve stopnje
računalništvo in informatika

Mentor: izr. prof. dr. Zoran Bosnić

Ljubljana, 2014

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

Kandidat naj v diplomski nalogi opiše problematiko nove zakonodaje, ki zahteva seznanitev uporabnikov z uporabo spletnih piškotkov, in posledic, ki jih zakonodaja ima na uporabniško izkušnjo. Razvije naj vtičnika za brskalnika Chrome in Firefox, ki omogočata shranjevanje uporabnikovih odločitev o sprejemanju piškotkov. Preuči naj možnosti objave razvite programske opreme v javne zbirke vtičnikov in metodološko naj primerja razlike v razvoju za oba brskalnika.

Izjava o avtorstvu diplomskega dela

Spodaj podpisani Aleš Papler, z vpisno številko **63090186**, sem avtor diplomskega dela z naslovom:

Vtičnik za pomnjenje nastavitev sprejemanja piškotkov na spletu

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom izr. prof. dr. Zorana Bosnića,
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela,
- soglašam z javno objavo elektronske oblike diplomskega dela na svetovnem spletu preko univerzitetnega spletnega arhiva.

V Ljubljani, dne 15. septembra 2014 Podpis avtorja:

Zahvalil bi se svojemu mentorju izr. prof. dr. Zoranu Bosniću za vse nasvete in korekten odnos pri izdelavi diplomske naloge. Hvaležen sem tudi staršem, ki so mi omogočili študij in me spodbujali v času študija.

Kazalo

Slike

Primeri

Seznam kratic in simbolov

ACID (angl. *Atomicity, Consistency, Isolation, Durability*) – sistem za upravljanje s podatkovnimi bazami mora zagotoviti lastnosti ACID (atomarnost, konsistentnost, izolacija in trajnost) za vsako transakcijo

AJAX (angl. *Asynchronous JavaScript and XML*) – skupina povezanih spletnih razvojnih tehnik, ki se uporabljajo za izdelavo interaktivnih spletnih strani in aplikacij. Spletne strani si s pomočjo AJAX-a v ozadju asinhrono izmenjujejo podatke s strežnikom

API (angl. *Application Programming Interface*) – aplikacijski programski vmesnik, ki določa, kako morajo posamezne komponente programske opreme sodelovati med seboj

DTD (angl. *Document Type Definition*) – označevalni jezik, ki določa pravila za strukturiranje dokumenta XML

HTTP (angl. *HyperText Transfer Protocol*) – aplikacijski protokol, ki služi za izmenjavo podatkov na spletu

RDF (angl. *Resource Description Framework*) – ogrodje za opisovanje virov in izmenjavo podatkov na spletu

SQL (angl. *Structured Query Language*) – strukturiran povpraševalni jezik za delo s podatkovnimi bazami

XML (angl. *Extensible Markup Language*) – označevalni jezik, ki določa format za zapis strukturiranih podatkov, ali arhitektura za prenos podatkov po omrežju

Povzetek

V okviru diplomskega dela sta bila razvita dva vtičnika za spletna brskalnika, ki preprečujeta pojavljanje obvestil o piškotkih na spletnih straneh. Osnova za razvoj vtičnikov je nova evropska zakonodaja, ki od upravljavcev spletnih strani zahteva, da na svoje strani postavijo vidno obvestilo o piškotkih. Vtičnika imata možnost pomnjenja uporabnikovih odločitev o sprejemanju ali zavrnitvi piškotkov na spletnih straneh. Ob ponovnem obisku strani in ob prisotnem obvestilu o piškotkih vtičnika samodejno izvedeta dejanje, ki ga je uporabnik pred tem izbral. V diplomskem delu je predstavljen postopek razvoja vtičnikov za spletna brskalnika Google Chrome in Mozilla Firefox ter opis različnih razvijalskih tehnik in pristopov. Delo vsebuje tudi prikaz postopka objave obeh vtičnikov v uradnih spletnih trgovinah brskalnikov.

Ključne besede: piškotki, vtičnik, brskalnik, Google Chrome, Mozilla Firefox

Abstract

Two web browser extensions that prevent cookie notifications on websites have been developed within this thesis. The motivation for developing the extensions is the new European legislation that requires website administrators to equip their sites with visible notifications that cookies are used within the website. The user's preferences of accepting or rejecting website cookies can be recalled by the developed extensions. On any subsequent visit to a website last user's action about accepting website notices is automatically carried out. The process of developing extensions for Google Chrome and Mozilla Firefox web browsers, as well as a description of different development techniques and approaches are presented in the thesis. A demonstration of publishing both extensions in the official web browsers' web stores is also shown in the thesis.

Keywords: cookies, extension, browser, Google Chrome, Mozilla Firefox

Poglavje 1

Uvod

Nova evropska direktiva 2009/136/ES, ki govori o zasebnosti in elektronskih komunikacijah, je z uveljavitvijo prinesla številne spremembe tudi na svetovnem spletu. Ena izmed vidnejših sprememb je obvezna uporaba obvestila o uporabi piškotkov na spletnih straneh. Upravljalci spletnih strani so po novem dolžni obvestiti obiskovalca, katere piškotke stran uporablja za delovanje in kakšen je njihov namen. Ravno ta obvestila pa so postala moteča pri vsakodnevnem brskanju po spletu. Direktiva predvideva tudi možnost, da uporabnik piškotke sprejme oziroma zavrne. Z vpeljavo te možnosti se pojavi nova težava v primeru, ko obiskovalec zavrne nameščanje piškotkov, vendar mu stran ob ponovnem obisku vseeno pokaže isto obvestilo.

Vse omenjene težave so bile motivacija za izdelavo mehanizma, ki bi uporabniku omogočal enostavnejše brskanje po spletu. Rešitev, ki smo si jo zamislili in razvili, je vtičnik za spletni brskalnik. Vtičnik na preprost način brskalniku doda nove funkcionalnosti, pri tem pa ne poslabša uporabniške izkušnje. To je bil po našem mnenju najprimernejši mehanizem za premostitev opisanih težav, zato smo ga razvili za brskalnika Google Chrome in Mozilla Firefox.

V drugem poglavju diplome bomo opisali spletne piškotke in nato na kratko povzeli novo evropsko ter slovensko zakonodajo s področja spleta in uporabe spletnih piškotkov. Tretje poglavje je v celotni namenjeno opisu razvoja vtičnika za brskalnik Google Chrome. V tem poglavju najprej opisujemo orodja in tehnologije, ki smo jih uporabili, na koncu pa tudi postopek objave vtičnika v uradni spletni trgovini. Sledi poglavje, namenjeno opisu razvoja vtičnika za brskalnik Mozilla

Firefox, njegova zgradba pa je podobna drugemu poglavju. V petem poglavju je opisano delovanje obeh vtičnikov in narejena medsebojna primerjava obeh razvojev. Zadnje poglavje povzema sklepne ugotovitve, do katerih smo prišli med razvojem obeh vtičnikov, in podaja predloge za izboljšave.

Poglavje 2

Zakonodaja in piškotki

2.1 Ozadje problema

Upravljalci spletnih strani so morali, kot že uvodoma omenjeno, ob uveljavitvi nove zakonodaje na svoje strani vključiti vse spremembe, ki jih zakonodaja predvideva. Ena izmed vidnejših sprememb je prikaz obvestila uporabnikom spletnih strani o piškotkih, ki se nahajajo na strani. Več o spremembah zakonodaje smo napisali v poglavju 2.3.

Težav pri opisani zahtevi je več. Prva težava, ki se ob tem pojavi, je v primeru, ko uporabnik ob prvem obisku strani jasno pove, da ne želi nameščanja piškotkov na njegovo strojno opremo (računalnik, pametni telefon ali tablični računalnik), ob vnovičnem obisku pa bo stran uporabniku ponovno prikazala isto obvestilo in še enkrat bo moral izraziti svojo odločitev glede nameščanja piškotkov. Druga težava je, da je obvestilo prikazano čez večino ali kar čez celotno stran v kombinaciji z domnevno privolitvijo uporabnika. Ob vsakem ponovnem obisku strani bo uporabnik naletel na to obvestilo in ga nato moral ročno zapreti. Tretja težava pa se pojavi, ko uporabnik v brskalniku izbriše celotno zgodovino brskanja. Ker si spletne strani odločitve uporabnika shranijo kot piškotke, se ob brisanju zgodovine brskanja vse te odločitve izgubijo. Strani bodo ob ponovnem obisku od uporabnika znova želele njegovo privolitev o nameščanju piškotkov.

Vse te težave so bile motivacija k razmišljanju, kako bi to omilili, če že ne povsem odpravili. Rešitev, ki smo se je domislili, je podrobnejše opisana v po-

glavju 2.4.

2.2 Piškotki

Piškotek (angl. *cookie*) je preprosta tekstovna datoteka, ki je sestavljena iz enega ali več parov ključ-vrednost (angl. *key-value*) [15]. Ob obisku spletne strani strežnik naloži enega ali več piškotkov na uporabnikovo strojno opremo. Piškotek običajno vsebuje osnovne informacije o strežniku, času veljavnosti in enoličnem identifikatorju uporabnika. Piškotek sam po sebi ni škodljiv, saj ne more vsebovati nič drugega kot samo alfanumerične znake. Glavni nameni piškotkov so identifikacija uporabnikov, personalizacija videza spletnih strani in spletnih oglasov. Vendar pa lahko vsebuje tudi osebne podatke uporabnika. To se zgodi v primeru, ko uporabnik sam vnese osebne podatke v spletni obrazec in jih nato strežnik po prejemu obrazca shrani v piškotek in ga namesti na uporabnikov računalnik. Piškotek, ki vsebuje osebne podatke, pa predstavlja možno grožnjo za vdor v zasebnost.

Slaba stran piškotkov je, da lahko preko njih sledimo uporabniku preko različnih spletnih strani in si tako ustvarimo profil uporabnika – katere strani je obiskal, kaj mu je všeč in kakšne članke prebira. Ker pa se tako sledenje brez dovoljenja uporabnika šteje za vdor v uporabnikovo zasebnost, želi nova evropska zakonodaja to početje omejiti.

2.2.1 Zgradba piškotkov

Piškotek je zgrajen iz sedmih atributov [4]:

- ime piškotka,
- vrednost piškotka,
- čas veljavnosti,
- pot, za katero je namenjen piškotek,
- domena, za katero je namenjen piškotek,
- zastavica, če je potrebna varna povezava za dostop in uporabo piškotka,

- zastavica, če je dostop do piškotka možen le preko protokola HTTP ali pa je do njega možno dostopati tudi preko drugih protokolov (npr. preko AJAX-a).

Prva dva atributa piškotka sta obvezna in morata biti definirana, sicer piškotek ni veljaven.

2.2.2 Vrste piškotkov

- Sejni piškoti (angl. *session cookies*) so piškotki, ki jih strežnik izdela ob začetku seje in nimajo predpisanega časa veljavnosti, saj jih brskalnik izbriše po preteku seje ali ko uporabnik zapre brskalnik [7]. Običajno se v njih hrani podatek, katere podstrani je uporabnik na strani obiskal, katere izdelke je uporabnik dodal v košarico in podobno. Nahajajo se le v bralno-pisalnem pomnilniku uporabnikove strojne opreme in niso shranjeni na obstojnem mediju (npr. na trdem disku).

- Trajni piškotki (angl. *persistent cookies*) so piškotki, ki imajo čas veljavnosti daljši, kot je trajanje ene seje. To je lahko nekaj dni, mesecev ali celo več let. Shranjeni so na obstojnem mediju uporabnikove strojne opreme. V njih se hrani informacija o strani, ki jo je uporabnik obiskal, preden je prišel na trenutno stran, kolikokrat je stran že obiskal in kakšna je njegova jezikovna nastavitev strani. Brskalnik strežniku ob vsaki poslani zahtevi pošlje tudi trajni piškotek, če ta že obstaja za domeno in pot strani, ki si jo uporabnik trenutno ogleduje. Na ta način lahko strežnik analizira promet in gradi profile uporabnikov.

- Lastni piškotki (angl. *first-party cookies*) so piškotki izdelani s strani strežnika, ki gostuje spletno stran in imajo v domeni vpisano enako domeno, kot je domena spletne strani, ki si jo uporabnik trenutno ogleduje. *Primer:* če uporabnik obišče stran *www.primer.si* in se na njegov računalnik namestita dva piškotka; prvi z domeno *primer.si*, drugi pa z domeno *spletna.analitika.si*, prvi ustreza definiciji lastnega piškotka, drugi pa tujega piškotka. Lastni piškotek je lahko sejni ali trajni.

- Tuji piškotki (angl. *third-party cookies*) so piškotki, ki imajo v domeni vpi-

sano drugo domeno, kot je domena spletne strani, ki si jo uporabnik trenutno ogleduje. To so običajno spletne oglaševalske agencije in agencije za analitiko spletnega prometa. Ker se preko tujih piškotkov lahko sledi uporabnikovemu brskanju po spletu, se uporaba teh piškotkov brez dovoljenja uporabnika šteje kot vdor v uporabnikovo zasebnost.

2.3 Zakonodaja

V začetku leta 2013 je v veljavo stopila sprememba Zakona o elektronskih komunikacijah (Uradni list št. 109/2012), ki govori o novih pravilih in smernicah glede uporabe piškotkov in podobnih spletnih tehnologij za shranjevanje in pridobivanje podatkov, shranjenih na uporabnikovi strojni opremi. S to spremembo zakona se je v pravni red Republike Slovenije prenesla tudi Direktiva 2002/58/ES Evropskega parlamenta in Sveta z dne 12. julija 2002 o obdelavi osebnih podatkov in varstvu zasebnosti na področju elektronskih komunikacij (Direktiva o zasebnosti in elektronskih komunikacijah) [8], zadnjič spremenjena z Direktivo 2009/136/ES.

Pomembna novost, ki jo Direktiva vnaša v evropski in posledično tudi v slovenski spletni prostor je, da je ≫*shranjevanje podatkov ali pridobivanje dostopa do podatkov, shranjenih v terminalski opremi naročnika ali uporabnika, dovoljeno samo pod pogojem, da je zadevni naročnik ali uporabnik v to privolil po tem, ko je bil jasno in izčrpno obveščen v skladu z Direktivo 95/46/ES, med drugim o namenih obdelave.*≪ [6]

Kar pomeni, da je uporaba piškotkov po novem mogoča le, če spletna stran poda uporabniku zadostne informacije o vrsti, času veljavnosti in namenu piškotkov, ter ob privoliti uporabnika.

Glavni nameni omenjene direktive so torej:

- uporabniku zagotoviti pravico do zasebnosti in zaupnosti na spletu,
- predstaviti uporabniku splošne in podrobnejše informacije glede piškotkov, ki se bodo ob obisku določene spletne strani namestili na njegovo strojno opremo, in
- omogočiti uporabniku izbiro, da za določeno spletno stran sprejme oziroma zavrne nameščanje piškotkov na njegovo strojno opremo.

2.3.1 Privolitev

Spletna stran mora pred namestitvijo piškotkov, ki ne spadajo pod izjeme (opisane so v poglavju 2.3.2), dobiti od uporabnika privolitev. Pred tem naj bi uporabnik od spletne strani dobil vse zadostne informacije o piškotkih (o vrsti, času veljavnosti in namenu piškotkov). Poznamo dve vrsti soglasja:

- Pri domnevni privolitvi spletna stran na svoji začetni strani uporabniku prikaže obvestilo o piškotkih. V obvestilu je običajno zapisano le, da se z uporabo njihovih storitev ali z brskanjem po podstraneh sklepa, da je uporabnik podal privolitev o uporabi piškotkov.

- Za izrecno privolitev štejemo zavestno uporabnikovo odločitev o uporabi piškotkov po tem, ko je bil seznanjen z zadostnimi informacijami o uporabi in namenu piškotkov. To je običajno klik na gumb ali povezavo za strinjanje s pogoji uporabe. Spletna stran uporabniku tudi ponudi možnost, da se premisli o svoji odločitvi o uporabi piškotkov tudi po tem, ko je enkrat že izrazil svojo odločitev.

2.3.2 Izjeme

Zakon o elektronskih komunikacijah pa v drugem odstavku 157. člena predvideva tudi dve izjemi. Prva izjema govori o tem, da je dovoljena uporaba piškotkov oz. podobnih tehnologij izključno zaradi prenosa sporočila po omrežju. To pomeni, da prenos sporočila tehnično ne bi bil mogoč brez uporabe piškotkov. Druga izjema pa dovoljuje piškotke v primeru, da brez njih ne bi bilo mogoče zagotoviti uporabe storitve, ki jo je uporabnik izrecno zahteval.

Dodaten kriterij, ki mu morajo piškotki zadostiti, pa je njihovo trajanje. Namreč, da nek piškotek lahko spada pod izjeme, mora njegov čas veljavnosti trajati le toliko časa, kot je potrebno za dosego namena (npr. prenos sporočila do strežnika). Nekaj splošnih izjem je Delovna skupina iz člena 29 (angl. *Article 29 Working Party*)[1] opredelila v svojem mnenju 04/2012, ki govori o izjemah za potrditev soglasja piškotkov [5]:

[1] Dostopno na spletnem naslovu http://ec.europa.eu/justice/data-protection/article-29/index_en.htm.

- Lastni sejni piškotki, ki so namenjeni le shranjevanju enoličnega identifikatorja uporabnikove seje. Taki piškotki si lahko zapomnijo tudi uporabnikove vnose v spletne obrazce ali vsebino spletne košarice.

- Piškotki za avtentifikacijo uporabnika, ki so namenjeni temu, da neka spletna stran prepozna, ali je nek uporabnik že prijavljen ali ne. Na ta način se prijavljenemu uporabniku ni treba ponovno prijaviti na vsaki podstrani. Piškotki za avtentifikacijo uporabnika spadajo pod izjeme le v primeru, če so piškotki sejni in ne trajni (definicija je opisana v poglavju 2.2.2). V nasprotnem primeru pa ne spadajo več pod izjemo in je za njihovo uporabo treba pridobiti privolitev uporabnika.

- Piškotki, namenjeni za zagotavljanje uporabnikove varnosti. Namenjeni so preprečevanju napačnih vnosov in prijav na spletnih straneh. Njihov čas veljavnosti je prav zaradi njihovega namena običajno daljši.

- Sejni piškotki multimedijskih predvajalnikov, ki so namenjeni shranjevanju tehničnih podatkov o predvajani vsebini za potrebe predvajalnika (npr. Adobe Flash Player).

- Sejni piškotki, s katerimi se uravnava obremenitev strežnikov (angl. *server load balancing*). Taki piškotki hranijo podatke o končnem strežniku, ki je dodeljen uporabniku za potrebe prenosa sporočil.

- Piškotki za personalizacijo videza strani. Namenjeni so shranjevanju uporabnikovih nastavitev strani, kot so na primer jezik strani, razvrščanje ali filtriranje rezultatov na strani. Taki piškotki spadajo pod izjeme, če njihov čas veljavnosti traja le do konca seje.

- Piškotki, namenjeni vtičnikom družabnih omrežij. Ta omrežja omogočajo spletnim stranem, da vključijo njihovo vsebino na svojo stran (t.i. vtičniki). Njihov glavni namen je uporabnikom, ki so prijavljeni v družabno omrežje, omogočiti, da ob obisku različnih spletnih strani lažje delijo vsebino, jo priporočijo prijateljem in podobno. Vendar ti piškotki spadajo pod izjemo le v primeru, ko je uporabnik član nekega družabnega omrežja in je aktivno prijavljen. V primeru, da uporabnik ni član družabnega omrežja ali pa da

se uporabnik odjavi oziroma izbriše iz družabnega omrežja, se taki piškotki brez dovoljenja ne smejo naložiti na uporabnikovo strojno opremo.

2.3.3 Piškotki, za katere je potrebna privolitev

Vrste piškotkov, za katere je potrebno pridobiti privolitev uporabnika:

- Piškotki, namenjeni vtičnikom družabnih omrežij, ki sledijo uporabnikom. Upravljalci spletnih strani pogosto na svojo stran vključijo vtičnike družabnih omrežij, kar v osnovi ne predstavlja nobene grožnje za uporabnikovo zasebnost. Težava pa nastane takrat, ko se ti piškotki uporabijo tudi za sledenje vseh uporabnikov (prijavljenih, neprijavljenih, članov in nečlanov družabnih omrežij) po spletu in so pogosto povezani s tujimi piškotki za namene oglaševanja, analitike in raziskave trga. Za uporabo piškotkov, namenjenih vtičnikom družabnih omrežij, ki sledijo uporabniku, je potrebna privolitev uporabnika.

- Tuji piškotki, namenjeni oglaševanju in trženju. Sem spada tudi spletno vedenjsko oglaševanje (angl. *online behavioural advertising*), ki je posebna oblika personaliziranega oglaševanja. Oglaševalske agencije sledijo uporabniku po spletu in ob tem vodijo statistiko, kakšne strani uporabnik obiskuje, kakšni izdelki so mu všeč in na katerih straneh preživi največ časa. Iz statistik nato izdelajo uporabnikov profil in uporabniku prikazujejo le zanj najbolj relevantne oglase.

- Lastni piškotki za analitiko. Z uporabo teh piškotkov upravljalci spletnih strani štejejo promet na strani, ugotovijo, s katere strani je uporabnik prišel, in opazijo morebitne težave z navigacijo na strani. Ker taki piškotki niso nujno potrebni za delovanje, je njihova uporaba pogojena s privolitvijo uporabnika.

Rok za implementacijo sprememb je bil 15. junij 2013. Za nadzor nad izvajanjem novi odločb o piškotkih je pristojen informacijski pooblaščenec. 157. člen Zakona o elektronskih komunikacijah predvideva tudi denarne kazni za kršitelje 157. člena. Višina kazni je opredeljena v 235. členu istega zakona.

2.4 Opis rešitve

Rešitev, ki smo se je domislili, je vtičnik za brskalnik, ki omogoča uporabniku, da ob obisku spletne strani samodejno skrije obvestilo o piškotkih ali samodejno izvrši neko dejanje uporabnika (npr. klik na gumb ali zapiranje pojavnega okna).

Glede na to, da sta v Evropi in v Sloveniji najbolj priljubljena brskalnika Google Chrome in Mozilla Firefox [22], smo se odločili, da razvijemo vtičnik za omenjena brskalnika.

Cilji, ki smo si jih pri tem zadali, so naslednji:

- razviti vtičnik za brskalnika Google Chrome in Mozilla Firefox,
- uporabniku omogočiti čim bolj transparentno in preprostejše delo pri brskanju na spletu in sočasnem delovanju vtičnika,
- trajno shraniti odločitve uporabnika z namenom, da uporabniku po brisanju zgodovine brskanja ne bo treba ponovno privoliti ali zavrniti nameščanja piškotkov na njegovo opremo,
- uporabniku ponuditi možnost urejanja shranjenih odločitev.
- narediti medsebojno primerjavo razvoja obeh vtičnikov, in
- objaviti vtičnika v brskalnikovih spletnih trgovinah (*Chrome Web Store* za brskalnik Google Chrome in *Mozilla Add-ons* za brskalnik Mozilla Firefox).

Poglavje 3

Razvoj vtičnika za brskalnik Chrome

Brskalnik Google Chrome (v nadaljevanju brskalnik Chrome) je brezplačen spletni brskalnik, ki ga je razvilo podjetje Google [14]. Prva beta različica je bila za operacijski sistem Windows na voljo 2. septembra leta 2008 v 43 jezikih, prva stabilna različica pa 11. decembra istega leta. Prva uradna različica za operacijska sistema OS X in Linux je bila na voljo 25. maja 2010. Najnovejša različica brskalnika Chrome je 36.0 (na dan 31. julija 2014) in je na voljo v 53 svetovnih jezikih. Razvijalci so pri brskalniku Chrome ubrali povsem nov pristop. Razvoja so se namreč lotili z mislijo, da brskalnik ne bo omogočal le brskanja po preprostih spletnih straneh, temveč bo omogočal tudi preprosto pošiljanje e-pošte, spletno nakupovanje, plačevanje računov in izvajanje spletnih aplikacij. Razvili so zmogljiv brskalnik s preprosto oblikovanim uporabniškim vmesnikom.

Razvijanje vtičnika za brskalnik Chrome smo izvedli v programskem jeziku JavaScript. Za prikaz strani opcij in pojavnih oken pa potrebujemo še označevalni jezik HTML in slogovni jezik CSS.

3.1 Uporabljena orodja in tehnologije

Visual Studio

Za izdelavo vtičnika za brskalnik Chrome smo uporabili integrirano razvojno okolje (angl. *integrated development environment*) Visual Studio 2013, ki ga je razvilo podjetje Microsoft [26]. Visual Studio je orodje, namenjeno razvijalcem programske opreme, in se uporablja za razvoj namiznih aplikacij, spletnih aplikacij in spletnih storitev. Nudi podporo programskim jezikom C, C++, Visual Basic, Visual C#, F#, HTML, XML, CSS in JavaScript. Urejevalnik kode ima podporo za *IntelliSense* – to je orodje, ki pohitri proces pisanje kode. Ena izmed lastnosti orodja IntelliSense je tudi samodokončevanje kode (angl. *code completion*). Prav zaradi te lastnosti in zaradi dobre podpore JavaScriptu smo za razvoj vtičnika odločili za Visual Studio 2013.

JavaScript

JavaScript je skriptni programski jezik, ki se ga najpogosteje uporablja na spletnih straneh in pri spletnih aplikacijah. Omogoča spreminjanje in interakcijo programa s spletnimi stranmi, ne da bi bilo pri tem potrebno strani ponovno osvežiti oziroma naložiti.

CSS

CSS (angl. *Cascading Style Sheets*) je slogovni jezik, namenjen opisovanju videza spletnih dokumentov, napisanih v označevalnem jeziku (npr. HTML).

JSON

JSON (angl. *JavaScript Object Notation*) je tekstovna datoteka, ki vsebuje podatkovne objekte ter njihove lastnosti in vrednosti, sestavljene iz parov ključ-vrednost. Uporablja se pri prenosu podatkov po omrežju, najpogosteje v smeri od strežnika do spletne aplikacije.

HTML DOM

HTML DOM (angl. *Document Object Model*) je objektni model za predstavitve dokumentov HTML in XML, zgrajen v obliki drevesne strukture. Spletna stran je v brskalniku predstavljena v obliki drevesne strukture, z začetnim vozliščem imenovanim *Document object*.

3.2 Zgradba in sestava vtičnika

Razvoj vtičnika smo začeli tako, da smo najprej izdelali namensko mapo, v kateri se nahajajo vse potrebne podmape in datoteke. Da bo vtičnik pravilno deloval, je pomembna pravilna hierarhija podmap in datotek, znotraj glavne mape vtičnika. Obvezna je datoteka *manifest.json* in ena ali več datotek HTML. Opcijsko vtičnik lahko vsebuje tudi eno ali več datotek JavaScript in datoteke ostalih vrst (npr. slikovne datoteke, datoteke CSS, datoteke JSON za lokalizacijo vtičnika). Pri razvoju našega vtičnika smo uporabili več datotek HTML in JavaScript ter datoteki JSON, namenjeni lokalizaciji vtičnika. Na sliki 3.1 je prikazana struktura korenske mape našega vtičnika z vsemi datotekami, ki smo jih med razvojem potrebovali.

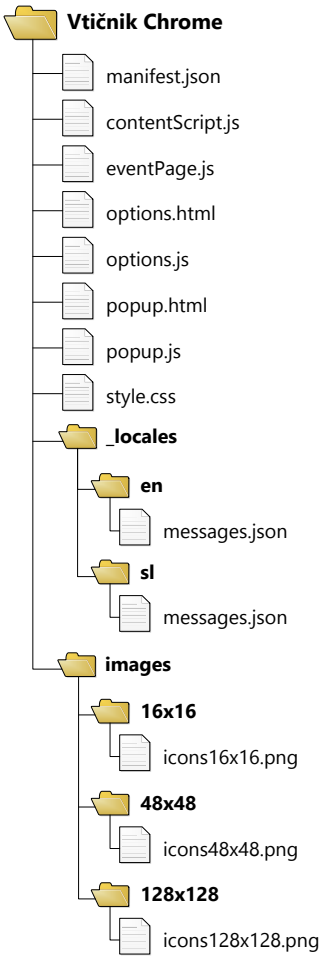
manifest.json

Vsak vtičnik ali aplikacija za brskalnik Chrome mora imeti eno datoteko JSON poimenovano *manifest.json*, ki vsebuje več pomembnih informacij za brskalnik [19]. Na primeru 3.1 je prikazana vsebina datoteke *manifest.json* iz našega vtičnika.

Prvi trije atributi (*manifest_version*, *name* in *version*) so obvezni, ostali so izbirni. Podrobnejši opis atributov:

Atribut ***manifest_version*** mora imeti vrednost 2 v primeru, da razvijamo vtičnik za brskalnik Chrome različice 18 ali višje. V nasprotnem primeru pa ima atribut vrednost 1.

Z atributom ***name*** poimenujemo vtičnik, z atributom ***description*** pa podamo kratek opis vtičnika. V primeru, da bi želeli lokalizirati ime in opis vtičnika, pa pravo vrednost nadomestimo z vrednostma *__MSG_extName__* in *__MSG_extDesc__*. Lokalizirane vrednosti hranimo v ločeni datoteki poimenovani *messages.json*, za vsak jezik posebej. Več o lokalizaciji smo napisali v poglavju 3.4.



Slika 3.1: Struktura korenske mape vtičnika za brskalnik Chrome

Primer 3.1: Vsebina datoteke chrome.manifest

```
{
    "manifest_version": 2,
    "name": "__MSG_extName__",
    "version": "1.0",
    "default_locale": "en",
    "description": "__MSG_extDesc__",
    "icons": {
            "16": "images/16x16/icon.png",
            "48": "images/48x48/icon.png",
            "128": "images/128x128/icon.png"
    },
```

```
    "browser_action": {
            "default_icon": "images/16x16/icon.png",
            "default_popup": "popup.html"
    },
    "background": {
            "scripts": ["background.js"],
            "persistent": false
    },
    "content_scripts": [
            {
                    "matches": ["*://*/*"],
                    "js": ["contentScript.js"]
            }
    ],
    "options_page":  "options.html",
    "permissions": [
            "storage",
            "notifications",
            "contextMenus",
            "tabs",
            "*://*/*"
    ]
}
```

Z atributom ***version*** določimo različico vtičnika. Ta atribut je pomemben pri samodejnemu posodabljanju vtičnika.

default_locale pove, katera lokalizacija je privzeta. Atribut je obvezen v primeru, če vtičnik vsebuje mapo *_locales*. V primeru, da vtičnik nima lokalizacije, ta atribut ne sme biti definiran.

Atribut ***icons*** pove brskalniku, kje se nahajajo ikone različnih velikosti, ki jih vtičnik uporablja.

Z atributom ***browser_action*** določimo vrsto vtičnika, v našem primeru gre za vtičnik, ki bo na voljo na vseh spletnih straneh. Več o vrstah vtičnika smo napisali v poglavju 3.3.2.

Atribut ***background*** pove brskalniku, da vtičnik vsebuje tudi skripto, ki se bo odzivala na dogodke in se izvajala v ozadju med uporabnikovim brskanjem. Naša skripta *eventPage.js* vsebuje poslušalce dogodkov, ki se odzivajo na dogodke uporabnika in na dogodke, ki jih prožijo druge skripte.

Z atributom ***content_scripts*** brskalniku povemo, da vtičnik vsebuje skripto, ki se bo izvedla ob prikazu neke spletne strani. Z dodatnim atributom *matches* povemo brskalniku, na katere spletne strani naj se ta skripta odziva. Zaradi var-

nostnih razlogov je to edina skripta znotraj vtičnika, ki ima omogočen dostop do spletnih strani DOM-a.

Z atributom ***options_page*** določimo stran, na kateri bo lahko uporabnik spreminjal nastavitve in videz vtičnika.

Atribut ***permissions*** vtičniku omogoči dostop do aplikacijskega programskega vmesnika brskalnika. Za naš vtičnik smo potrebovali dostop do brskalnikove hrambe podatkov (angl. *browser storage*), obvestil, menija desnega klika (angl. *context menu*) in brskalnikovega sistema za upravljanje z zavihki in okni. Več o programskih vmesnikih smo zapisali v poglavju 3.3.4.

eventPage.js

Vtičnik lahko vsebuje tako imenovano stran v ozadju (angl. *background page*), s pomočjo katere se vtičnik odziva na dogodke, doda nove funkcionalnosti brskalniku in podobno. Ločimo dve vrsti strani v ozadju – stran je lahko obstojna (angl. *persistent*) in je vedno aktivna, kar pomeni, da ne glede na to, ali uporabnik uporablja vtičnik ali ne, bodo stran in njene skripte vedno naložene v pomnilniku računalnika. Taka stran se imenuje *background page* in se pri izdelavi vtičnikov ne priporoča, saj tudi ob neaktivnosti zavzema vire računalnika. Veliko boljša rešitev je stran, ki je aktivna le takrat, ko se jo zares potrebuje. Taki strani pravimo *event page*. Stran običajno vsebuje poslušalce dogodkov (angl. *event listeners*), na katere se potem tudi ustrezno odzove z rokovalniki dogodkov (angl. *event handlers*).

Za potrebe našega vtičnika smo izbrali tip strani *event page*. Ta stran oziroma bolj natančno skripta JavaScript vsebuje poslušalce dogodkov. Eden izmed dogodkov je namestitev vtičnika, ko brskalniku dodamo dva nova elementa menija desnega klika. Ali pa dogodek, ko se spletna stran v celoti naloži in nato krmilnik tega dogodka preveri, če je uporabnik dodal vnos, da se izvede kakšno dejanje na tej spletni strani (samodejni klik ali samodejno skrivanje nekega spletnega elementa). Več o samem delovanju vtičnika smo opisali v poglavju 5.1.

Ostale skripte imajo možnost dostopati do funkcij strani v ozadju. To je možno preko funkcije *getBackgroundPage* vmesnika *chrome.runtime*. Primer 3.2 vsebuje klic funkcije strani v ozadju, ki ga izvedemo iz druge skripte. V funkciji povratnega klica uporabimo objekt JavaScript strani v ozadju (objekt *eventPage*) in nato preko tega objekta kličemo funkcijo.

Primer 3.2: Klic funkcije strani v ozadju iz drugih skript

```
chrome.runtime.getBackgroundPage(function (eventPage) {
        eventPage.addEntryToStorage("setting", {
                "name": "generalSwitch",
                "value": "disabled"
        });
});
```

Primer 3.2 je izsek kode iz našega vtičnika, ko želimo v skripti *options.js* spremeniti vrednost glavnega stikala v brskalnikovi hrambi podatkov preko funkcije strani v ozadju. Prvi parameter v funkciji *addEntryToStorage* vsebuje niz, s katerim povemo, da gre za nastavitev, drugi parameter pa je objekt JSON, ki vsebuje nove vrednosti glavnega stikala.

contentScript.js

Content script je datoteka JavaScript, ki deluje v obsegu spletnih strani. Z uporabo HTML DOM-a lahko *content script* bere vsebino in tudi spreminja videz spletnih strani. Zaradi varnostnih razlogov pa je to tudi edina datoteka znotraj vtičnika, ki ima neposreden dostop do spletnih strani in njihove vsebine.

Ima pa tudi nekaj omejitev, na katere je potrebno paziti pri razvoju vtičnika. Namreč, *content script* ne more uporabljati vseh funkcij programskega vmesnika *chrome*, razen nekaterih izjem[1], in ne more uporabljati spremenljivk in funkcij spletnih strani ali vtičnika.

V *content scriptu* našega vtičnika se nahaja le funkcija za pridobivanje lastnosti spletnih elementov (ID, ime in razred elementa) in za samodejno izvršitev dejanja. Več o samem delovanju vtičnika smo napisali v poglavju 5.1.

style.css

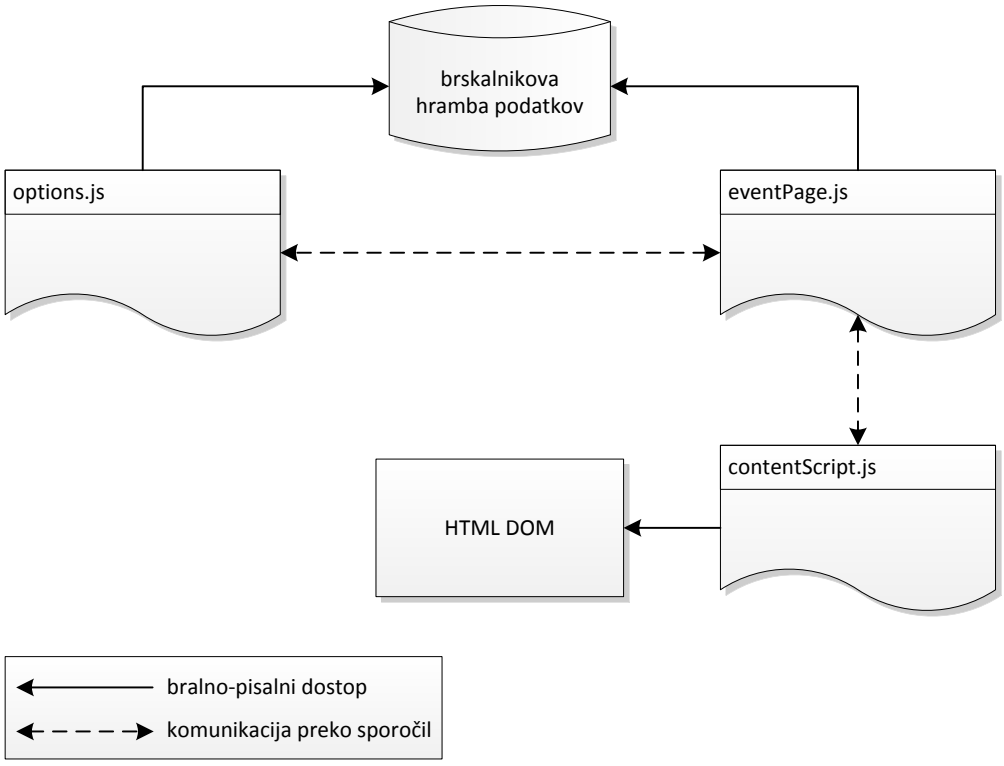
Za oblikovanje videza opcijske strani vtičnika (*options.html*) in pojavnega okna (*popup.html*) smo uporabil slogovni jezik CSS. Pri tem smo uporabili le nekaj preprostih slogovnih pravil, kot so na primer barva besedila, pozicioniranje elementov HTML in oblikovanje tabele HTML.

[1] Programski vmesniki, ki jih skripta *content script* lahko uporablja, so *chrome.i18n*, *chrome.storage*, *chrome.runtime* in *chrome.extension*.

3.3 Uporabljene tehnike in pristopi

3.3.1 Programska struktura vtičnika

Na sliki 3.2 je prikazan diagram programskih komponent za naš vtičnik in odnosi med posameznimi komponentami. Bralno-pisalni dostop do brskalnikove hrambe podatkov imata skripti *options.js* in *eventPage.js* preko funkcij aplikacijskega vmesnika *chrome.storage*. Skripte lahko med seboj tudi komunicirajo. Njihova komunikacija poteka preko sporočil, kar bomo podrobneje opisali v poglavju 3.3.5. V našem primeru med seboj komunicirata skripti *options.js* in *eventPage.js* ter skripti *eventPage.js* in *contentScript.js*. Kot že omenjeno, ima le skripta *contentScript* neposreden (bralno-pisalni) dostop do spletne strani HTML DOM-a.



Slika 3.2: Diagram komponent vtičnika za brskalnik Chrome

3.3.2 Browser action, page action

Pri razvoju vtičnika za brskalnik Chrome imamo na izbiro dve možnosti. Prva možnost je, da za vrsto vtičnika izberemo ***browser action***. Tak vtičnik se bo odzival na akcije brskalnika in bo na voljo na vseh spletnih straneh. Ikona vtičnika bo prikazana v desnem delu brskalnikove orodne vrstice. Vtičniku lahko dodamo še stran z možnostmi (angl. *options page*), na kateri bo lahko uporabnik spreminjal nastavitve in videz vtičnika.

Druga možnost je, da za vrsto vtičnika izberemo ***page action***. To pomeni, da bo vtičnik na voljo le, ko se naložijo določene spletne strani. Seznam spletnih strani, pri katerih bo vtičnik na voljo, določimo v datoteki *manifest.json*.

Za potrebe našega vtičnika smo izbrali *browser action*, ker smo želeli, da bo vtičnik na voljo pri vseh spletnih straneh.

3.3.3 Sinhroni in asinhroni način

V JavaScriptu in tudi v nekaterih drugih programskih jezikih obstajata dve vrsti načina izvajanja programa – sinhroni in asinhroni način [1]. Pri **sinhronem načinu** se program in klici funkcij izvajajo zaporedno, torej eden za drugim. Naslednji klic se ne bo pričel izvajati, dokler prejšnji klic ni v celoti dokončan. Uporaba sinhronega načina zadošča, če je program preprost in nima nobenega povezovanja preko mreže, bodisi s strežnikom ali z drugimi klienti.

Asinhroni način je priporočljiv v primeru, da želimo v programu izvesti poizvedbo SQL in nato prikazati rezultate te poizvedbe. Ali pa v primeru, ko na strežnik pošljemo nek zahtevek in glede na odgovor nato nadaljujemo izvajanje našega programa. V asinhronem načinu čakanje na rezultate poizvedbe ali na odgovor strežnika ne bo blokiralo oziroma ustavilo izvajanja primarnega programa. Vendar to velja le v primeru, da je nadaljnje izvajanje programa neodvisno od rezultatov poizvedbe ali od odgovora strežnika.

Prikaz rezultatov poizvedbe oziroma nadaljnje izvajanje programa glede na odgovor strežnika izvedemo v tako imenovani funkciji povratnega klica (angl. *callback function*). Ta funkcija se bo pričela izvajati, ko se bo asinhron klic zaključil in vrnil rezultate.

Preprost primer sinhronega izvajanja je prikazan na primeru 3.3, ko najprej

pridobimo zahtevo in jo nato sinhrono pošljemo na strežnik. Izvajanje celotnega programa se bo začasno zaustavilo, vse dokler program ne bo prejel odgovora strežnika in ga nato izpisal v konzolo. Izvajanje programa se bo nadaljevalo šele po uspešno prejetem odgovoru.

Primer 3.3: Primer sinhronega izvajanja

```
var zahteva = pripravi_zahtevo();
var odgovor = poslji_zahtevo_sinhrono(zahteva);
console.log(odgovor);
```

Primer 3.4 pa namesto sinhronega pošiljanja zahteve na strežnik vsebuje asinhrono pošiljanje. V tem primeru se bo izvajanje glavnega programa nadaljevalo, ko pa bo program prejel odgovor strežnika, ga bo izpisal v konzolo.

Primer 3.4: Primer asinhronega izvajanja

```
var zahteva = pripravi_zahtevo();
poslji_zahtevo_asinhrono(zahteva, function(odgovor) {
        console.log(odgovor);
});
```

Klice vmesnikov *Chrome* moramo izvesti asinhrono, razen če ni v dokumentaciji posameznega vmesnika predvideno drugače. Običajno le najbolj preproste klice vmesnikov izvedemo sinhrono. Ena takšnih izjem je klic vmesnika za pridobivanje lokalizirane vrednosti *chrome.i18n.getMessage(imeVrednosti, opcijskiParametri)*.

3.3.4 Programski vmesniki Chrome

Pri razvoju vtičnika smo uporabili več aplikacijskih programskih vmesnikov Chrome (angl. *Chrome APIs*). Celoten nabor vmesnikov je mogoče zaslediti na uradni strani za podporo razvoju vtičnika [17]. Vsi vmesniki so na omenjeni strani dobro dokumentirani in vsebujejo tudi preproste primere uporabe. Kot že omenjeno v poglavju 3.3.3, pa moramo večino klicev programskih vmesnikov izvajati asinhrono.

Nekaj programskih vmesnikov, ki smo jih uporabili:

chrome.storage

Ta vmesnik je namenjen shranjevanju, branju in sledenju uporabniških podatkov [23]. Ponuja povsem enake možnosti kot brskalnikova lokalna hramba podatkov (angl. *local storage*), ki jo uporabljajo spletne strani in aplikacije. Razlog, zaradi katerega smo se odločili za uporabo vmesnika *chrome.storage* namesto lokalne hrambe podatkov, je predvsem v ta, da so podatki uporabnika dostopni tudi po tem, ko uporabnik izbriše zgodovino brskanja. Drugi razlog pa je ta, da ima lokalna hramba podatkov doseg le za eno spletno domeno. Kar pomeni, da lahko beremo in urejamo vsebino hrambe podatkov le za tisto stran in njene podstrani, ki jih ima uporabnik trenutno odprte. To pa predstavlja težavo, saj ne moremo dobiti in prikazati celotne vsebine lokalne hrambe podatkov, ne da bi pri tem uporabili slabe rešitve.

Dobri lastnosti hrambe podatkov *chrome.storage* v primerjavi z lokalno hrambo podatkov sta tudi ti, da bralno-pisalne operacije potekajo asinhrono (pri lokalni hrambi zaporedno oziroma sinhrono) in da so lahko podatki shranjeni kot objekti, kar pa za lokalno hrambo ne velja, saj podatke lahko hrani le v obliki nizov (angl. *strings*).

Celotno vsebino hrambe podatkov dobimo tako, da podamo funkciji *sync.get* vmesnika *chrome.storage* kot prvi parameter vrednost *null*. V primeru, da želimo prebrati le en objekt iz hrambe podatkov, pa funkciji podamo kot prvi parameter vrednost ključa želenega objekta. Primer 3.5 kaže, kako v funkciji povratnega klica izpišemo celotno vsebino hrambe podatkov in kako izpišemo le vsebino enega objekta.

Primer 3.5: Branje vsebine hrambe podatkov

```
// Izpis celotne vsebine hrambe podatkov
chrome.storage.sync.get(null, function(items){
        console.log(items);
});

// Izpis vsebine objekta CNP
chrome.storage.sync.get("CNP", function (items) {
        console.log(items)
});
```

Za shranjevanje v hrambo podatkov je namenjena funkcija *sync.set* vmesnika

chrome.storage. Kot prvi parameter funkciji podamo objekt, ki ga želimo shraniti. Drugi parameter, ki ni obvezen, je funkcija povratnega klica, znotraj katere obravnavamo uspešno ali neuspešno izvršitev shranjevanja. Objekt mora biti sestavljen iz parov ključ-vrednost. Za večjo urejenost znotraj hrambe podatkov in za lažje iskanje naj prvi ključ predstavlja ključ objekta, ki ga shranjujemo, vrednost tega ključa pa naj bo vsebina objekta, urejena po parih ključ-vrednost. V primeru 3.6 smo v hrambo podatkov shranili objekt, ki vsebuje primitivne tipe, polje in objekt.

Primer 3.6: Shranjevanje objekta v hrambo podatkov

```
chrome.storage.sync.set({ "testniObjekt": {
                kljuc1: ["abc", true, 123],
                kljuc2: 456,
                kljuc3: {
                        kljuc3-1: "vrednost3-1",
                        kljuc3-2: "vrednost3-2"
                }
        }
});
```

Pri tem smo opazili, da če želimo posodobiti le vrednost nekega ključa, npr. *kljuc2* v primeru 3.6 znotraj že shranjenega objekta, moramo v hrambo podatkov ponovno shraniti celoten objekt (*testniObjekt*), drugače se bodo ostale vrednosti znotraj objekta izgubile. To smo rešili na način, da smo objekt iz hrambe podatkov, preden smo želeli neko vrednost posodobiti, v celoti prebrali, ga posodobili in nato shranili nazaj.

chrome.notifications

Ta vmesnik je namenjen prikazu brskalnikovih obvestil uporabniku. Za potrebe našega vtičnika smo ta vmesnik potrebovali za obveščanje uporabnika o pomembnih dogodkih, ki so se zgodili. S klicem *chrome.notifications.create* izdelamo obvestilo, ki se bo uporabniku prikazalo v spodnjem desnem kotu zaslona. Primer klica vmesnika *chrome.notifications*:

Primer 3.7: Uporaba vmesnika za izdelavo obvestila

```
chrome.notifications.create("idObvestila", {
        type: "basic",
        title: "Naslov obvestila",
        message: "Vsebina obvestila."
```

```
}, function() { });
```

chrome.contextMenus

Vmesnik *chrome.contextMenus* je namenjen dodajanju novih elementov za meni desnega klika (angl. *context menu*) znotraj brskalnika. Pri našem vtičniku smo dodali dva elementa, kot je prikazano na sliki 5.2. Za izdelavo novega elementa podamo funkciji *chrome.contextMenus.create* kot prvi parameter objekt, v katerem so definirane lastnosti elementa, ki se bo pojavil v meniju desnega klika. Za primer 3.8 smo definirali napis, enolični identifikator in pogoje, pri katerih bo na voljo ta element.

Primer 3.8: Izdelava novega elementa za meni desnega klika

```
chrome.contextMenus.create({
        "title": "Nov element desnega klika",
        "id": "context-newElement",
        "contexts": [
                "page",
                "frame",
                "selection",
                "link",
                "editable",
                "image"
        ]
});
```

chrome.i18n

Vmesnik *chrome.i18n* je namenjen lokalizaciji vtičnika. Pri tem mora vtičnik vsebovati mapo *_locales* s toliko podmapami in datotekami *messages.json*, koliko različnih jezikov želimo. Na sliki 3.1 je prikazana hierarhija datotek in map znotraj glavne mape vtičnika. Več o sami lokalizaciji smo povedali v poglavju 3.4.

3.3.5 Komunikacija preko sporočil

Komunikacija med stranjo v ozadju (*eventPage.js*) in skripto *contentScript* ali med ostalimi skriptami lahko poteka preko sporočil [20]. Na eni strani je pošiljatelj sporočila, ki ob dogodkih ali dejanjih uporabnika pošlje sporočilo točno določene

vrste vsem poslušalcem. Na drugi strani je prisoten en ali več poslušalcev sporočil. V primeru, da je sporočilo prave vrste za nekega poslušalca, se ta poslušalec nanj odzove in opcijsko pošlje pošiljatelju tudi odgovor. Sporočilo in odgovor sta v obliki objekta JSON, torej lahko vsebujeta primitivne tipe, polje ali objekt.

Ker ima samo skripta *contentScript* dostop do modela HTML DOM, mi pa smo želeli dostopati do elementov DOM-a tudi v skripti *eventPage*, smo to težavo rešili ravno s komunikacijo preko sporočil.

Na primeru 3.9 je prikazana komunikacija preko sporočil med skriptama *eventPage.js* in *contentScript.js* v našem vtičniku.

Primer 3.9: Komunikacija preko sporočil med dvema skriptama

```
// Izsek programske kode iz skripte eventPage.js
var currentTab = getCurrentTab();
chrome.tabs.sendMessage(currentTab.id, {
        action: "getElementProperties"
}, function (response) {
        var elementID = response.id;
        var elementName = response.name;
        var elementClass = response.class;
});

// Izsek programske kode iz skripte contentScript.js
chrome.runtime.onMessage.addListener(
    function (request, sender, sendResponse) {
        if (request.action == "getElementProperties") {
            var element = getClickedElement();
            sendResponse({
                id: element.getAttribute("id"),
                name: element.getAttribute("name"),
                elClass: element.getAttribute("class")
            });
        }
});
```

Za komunikacijo preko sporočil smo uporabili funkcijo *sendMessage* vmesnika *chrome.tabs*. Funkcija sprejme kot prvi parameter identifikator trenutno odprtega zavihka, drugi parameter je sporočilo v obliki JSON, tretji parameter pa je funkcija povratnega klica, v kateri obravnavamo odgovor na sporočilo. Na poslušalčevi strani se nahaja poslušalec dogodkov, ki ob prejetju sporočila najprej preveri, če je sporočilo pravega tipa, nato pa preko funkcije *sendResponse* pošlje pošiljatelju odgovor – v našem primeru pošlje ID, ime in razred spletnega elementa, na katerega

je uporabnik kliknil.

3.3.6 Ikona orodne vrstice in značka

Vtičnik ima lahko v orodni vrstici brskalnika svojo ikono. Ob kliku nanjo se uporabniku prikaže pojavno okno (angl. *popup*), v katerem lahko prikažemo informacije o stanju vtičnika, brskalnika ali o trenutno prikazani spletni strani. Za demonstracijske potrebe smo v pojavnem oknu našega vtičnika prikazali povezavo, ki ob kliku odpre stran z možnostjo vtičnika. V datoteki *manifest.json* smo ikono v orodni vrstici in pojavno okno definirali tako:

Primer 3.10: Dodajanje ikone vtičnika v orodno vrstico brskalnika

```
"browser_action": {
            "default_icon": "images/16x16/icon.png",
            "default_popup": "popup.html"
}
```

Ikoni v orodni vrstici pa lahko dodamo tudi tako imenovano značko (angl. *badge*), ki prekrije spodnji del ikone s poljubnim besedilom. To storimo preko funkcije *setBadgeText* vmesnika *chrome.browserAction*. Za potrebe našega vtičnika smo ikoni dodali značko z besedilom *ON* oziroma *OFF*, ki se prikaže glede na stanje glavnega stikala vtičnika. Primer uporabe značke za ikono vtičnika:

Primer 3.11: Dodajanje značke ikoni v orodni vrstici brskalnika

```
chrome.browserAction.setBadgeText({ "text": "ON" });
```

3.4 Lokalizacija

Vtičnik lahko razvijemo na način, da so izrazi, elementi in obvestila vtičnika v istem jeziku, kot je jezik brskalnika. Da je vsebina vtičnika lokalizirana glede na jezik brskalnika, mora glavna mapa vtičnika vsebovati mapo *_locales*, znotraj katere je potem toliko podmap, v koliko jezikov želimo lokalizirati vtičnik. Podmape pa morajo biti poimenovane po veljavnih kraticah za jezike [16]. Za naš vtičnik smo naredili podporo za dva jezika – angleščino in slovenščino. Organizacija datotek in map znotraj vtičnika je prikazana na sliki 3.1. Na primeru 3.12 je prikazan tudi

del vsebine datoteke *messages.json* za angleški jezik, na primeru 3.13 pa enak del vsebine za slovenski jezik.

Primer 3.12: Del vsebine datoteke messages.json za angleški jezik

```
{
        "extName": {
                "message": "Cookie Notification Preventer"
        },
        "table_button_edit": {
                "message": "Edit entry"
        },
        "confirm_deleting_entry": {
                "message":
                        "Would you really like to delete the entry with Row
                            ID: $rowID$?",
                "placeholders": {
                        "rowID": {
                                "content": "$1",
                                "example": "1"
                        }
                }
        },
        ...
}
```

Primer 3.13: Del vsebine datoteke messages.json za slovenski jezik

```
{
        "extName": {
                "message":"Preprečevalec obvestil piškotkov"
        },
        "table_button_edit": {
                "message": "Uredi zapis"
        },
        "confirm_deleting_entry": {
                "message":
                        "Ali res želite izbrisati zapis z ID vrstice:
                            $rowID$?",
                "placeholders": {
                        "rowID": {
                                "content": "$1",
                                "example": "1"
                        }
                }
        },
        ...
```

```
}
```

Če želimo uporabiti lokalizirane vrednosti znotraj datoteke *manifest* ali datoteke CSS, to storimo tako, da imenu spremenljivke spredaj dodamo *__MSG_*. Primer uporabe lokalizirane vrednosti znotraj datoteke *manifest.json* je prikazan na primeru 3.1.

Za pridobivanje lokaliziranih vrednosti znotraj skript to storimo preko funkcije *getMessaage* vmesnika *chrome.i18n*. Primer uporabe lokaliziranih vrednosti v datoteki *options.js*:

Primer 3.14: Uporaba lokaliziranih vrednosti znotraj programske kode

```
var tableEditButtonText =
        chrome.i18n.getMessage("table_edit_button");

// Primer s parametrom
var rowId = 5;
var deletingEntryMsg = chrome.i18n.getMessage("confirm_deleting_entry",
                            [rowId]);
```

3.5 Lokalno nameščanje vtičnika

Vtičnik lahko namestimo lokalno, kar pomeni, da ga namestimo iz datotečnega sistema v naš brskalnik. Pri tem mora biti mapa vtičnika v podobni organizaciji kot na sliki 3.1. Vtičnik namestimo tako, da v meniju brskalnika poiščemo *Orodja* in izberemo *Razširitve* ali pa v naslovno vrstico brskalnika vpišemo *chrome://extensions*. Na strani razširitev najprej omogočimo način za razvijalce (angl. *developer mode)* in kliknemo na gumb *Naloži odpakirano razširitev* (angl. *Load unpacked extension*). V pojavnem oknu izberemo mapo vtičnika in v primeru, da vtičnik ne vsebuje nobene sintaktične napake, je namestitev končana.

Vtičnik pa lahko objavimo tudi v spletni trgovini *Chrome Web Store*. Nameščanje vtičnika preko trgovine je preprostejše, saj nam ni potrebno omogočiti načina za razvijalce, prav tako pa si lahko drugi uporabniki brskalnika Chrome na ta način namestijo naš vtičnik. Postopek nalaganja in objave v spletni trgovini bomo opisali v poglavju 3.7.1.

3.6 Razhroščevanje

Razhroščevanje (angl. *debugging*) je proces odkrivanja in zmanjševanja napak pri razvoju programske ali strojne opreme [10]. Pri tem procesu razvijalec opazuje, če se del programa oziroma celoten program ali strojna oprema odziva in deluje v skladu s pričakovanji. Pri tem je razvijalcu na voljo množica orodij za razhroščevanje.

Brskalnik Chrome že v osnovi vsebuje Orodja za razvijalce (angl. *Developer tools*), ki povsem zadostujejo razhroščevanju vtičnika. Orodja za razvijalce so dostopna preko menija brskalnika (*Orodja, Orodja za razvijalce*) ali preko kombinacije tipk *Ctrl+Shift+I*. Omogočajo pregled DOM-a, slogovnih pol CSS in skript JavaScript za spletne strani in vtičnike. Prav tako si lahko nastavimo prekinitveno točko (angl. *breakpoint*) za začasno prekinitev izvajanja skript JavaScript, si ogledamo vrednosti spremenljivk in v konzoli izvajamo kodo JavaScript.

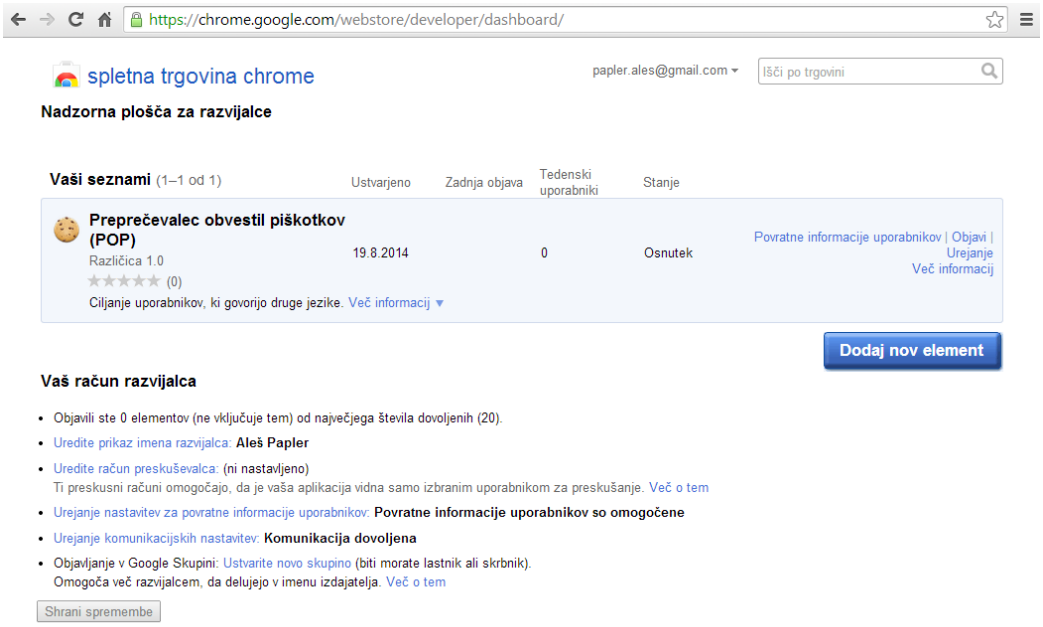
Pri razhroščevanju vtičnika pa je potrebno paziti na eno izjemo. Namreč, vtičnik je lahko sestavljen iz štirih delov – iz strani v ozadju, strani uporabniškega vmesnika (sem spada pojavno okno, ki se pojavi ob kliku na ikono vtičnika v orodni vrstici brskalnika), strani možnosti in skripte *content script*. Vsi ti deli pa imajo ločena orodja za razvijalce, kar pomeni, da je ob morebitni napaki vtičnika le-to treba iskati na štirih različnih mestih.

3.7 Spletna trgovina Chrome

Spletna trgovina Chrome (angl. *Chrome Web Store*) je spletno mesto, ki je namenjeno za objavo in prenos spletnih aplikacij za brskalnik Chrome ali za Googlove aplikacije (angl. *Google Apps*) [9]. Trgovina je bila ustanovljena 6. decembra 2010 in je dostopna na spletnem naslovu https://chrome.google.com/webstore. V trgovini je na voljo veliko brezplačnih in plačljivih aplikacij, razširitev (vtičnikov) in tem za brskalnik Chrome. Vsak izdelek v trgovini ima svojo stran, na kateri lahko najdemo dodatne informacije o njem ter preberemo ocene in mnenja uporabnikov.

3.7.1 Objava vtičnika v spletni trgovini

V tem poglavju smo opisali, kako smo objavili naš vtičnik v spletni trgovini Chrome. Postopek za objavo aplikacije ali teme za brskalnik Chrome je zelo podoben. Za objavo vtičnika v spletni trgovini Chrome smo najprej potrebovali Googlov uporabniški račun. Nato smo odprli nadzorno ploščo za razvijalce (angl. *developer dashboard*) [11] in preko nje dodali nov element – korensko mapo vtičnika, ki je morala biti stisnjena v formatu ZIP. Na sliki 3.3 je prikazana nadzorna plošča za razvijalce, po tem, ko smo naložili osnutek našega vtičnika.



Slika 3.3: Nadzorna plošča za razvijalce v spletni trgovini Chrome

Osnutku vtičnika smo nato dodali podrobnejši opis v angleščini in slovenščini, ikono, zaslonske posnetke delovanja vtičnika, ga umestili v eno izmed kategorij vtičnikov in mu nastavili vse ostale dodatne nastavitve (v katerih državah je vtičnik na voljo, ga nastavili tako, da je javno viden vsem in da je brezplačen). Pred končno objavo izdelka v trgovini je bilo treba Googlu plačati tudi enkratni prispevek za razvijalce v višini 5 dolarjev.

Končni videz vtičnika, objavljenega v spletni trgovini Chrome, prikazuje slika 3.4.

Vtičnik je dostopen na spletnem naslovu https://chrome.google.com/webstore/detail/cookie-notification-preve/nibhfnfgknipgjgipdoapppjpeakljek.

Slika 3.4: Objava vtičnika v spletni trgovini Chrome

Poglavje 4

Razvoj vtičnika za brskalnik Firefox

Brskalnik Mozilla Firefox (v nadaljevanju brskalnik Firefox) je odprtokodni in brezplačen spletni brskalnik [13]. Na voljo je za različne operacijske sisteme in platforme, kot so Windows, OS X, Linux in Android. Prva verzija brskalnika je bila na voljo že leta 2002 pod imenom Phoenix. Kasneje se je brskalnik zaradi sporov glede poimenovanja še dvakrat preimenoval, najprej v Firebird in na koncu še v Firefox. Trenutna verzija brskalnika je 32.0 (na dan 4. septembra 2014), na voljo pa je v več kot 80 jezikih. Brskalnik Firefox je po priljubljenosti v Evropi na drugem mestu (avgust 2014), večjo priljubljenost med uporabniki ima le brskalnik Chrome [22].

Za izdelavo vtičnika za brskalnik Firefox smo uporabili tako imenovani tradicionalni pristop. Pri tem pristopu razvijalec uporabi eno ali več prekrivnih datotek XUL, ki razširijo obstoječi uporabniški vmesnik brskalnika ter mu dodajo nove funkcionalnosti. Več o prekrivnih datotekah in pristopu bomo omenili v poglavju 4.3.1. Vtičnik smo razvili v programskem jeziku JavaScript, potrebovali pa smo še označevalni jezik HTML, slogovni jezik CSS in strukturiran povpraševalni jezik SQL.

4.1 Uporabljena orodja in tehnologije

4.1.1 Komodo Edit

Komodo Edit je brezplačno in odprtokodno integrirano razvojno okolje, ki ga je razvilo kanadsko podjetje ActiveState [18]. Je del naprednejšega in plačljivega razvojnega okolja Komodo IDE. Komodo Edit je namenjen pisanju in urejanju programske kode, napisane v enem izmed dinamičnih programskih jezikov (Python, PHP, Ruby, JavaScript) ter nudi podporo označevalnim jezikom (HTML, CSS, XML). Ker Komodo Edit temelji na platformi Mozilla XULRunner, nudi dobro podporo tudi pri razvoju vtičnikov za Firefox (npr. samodejno zaključevanje značk XUL). Pri razvoju vtičnika smo uporabili različico 8.5.3.

4.1.2 SQLite

Je relacijski sistem za upravljanje s podatkovno bazo (angl. *relational database management system*), ki za delovanje ne potrebuje namenskega strežnika [3]. Podatkovna baza in njen sistem za upravljanje sta preko ene same knjižnice direktno integrirana v aplikacijo, ki uporablja SQLIte. Vse transakcije SQLite so združljive z načeli ACID. V večini primerov se uporablja za lokalno hrambo podatkov. Celotna podatkovna baza, njene tabele in indeksi so združeni v eno datoteko.

Pri shranjevanju podatkov vtičnika smo se odločili za uporabo SQLita, saj naj bi bil po mnenju organizacije Mozzila to najprimernejši mehanizem za hrambo podatkov znotraj brskalnika. Brskalnik Firefox že uporablja SQLite za hrambo zaznamkov, piškotkov in zgodovine brskanja. Več o uporabi hrambe podatkov na primeru našega vtičnika smo napisali v poglavju 4.3.4.

4.1.3 SQLite Database Browser

Za delo z datotekami SQLite smo uporabili odprtokodno in brezplačno orodje SQLite Database Browser,[1] ki omogoča izdelavo, pregled in urejanje datotek SQLite. Orodje smo potrebovali predvsem pri razhroščevanju in testiranju poizvedb SQL, ki jih uporablja vtičnik pri hrambi podatkov.

[1] Od verzije 3.3.1 naprej se orodje imenuje Database Browser for SQLite.

4.1.4 XUL

XUL (angl. *XML User Interface Language*) je označevalni jezik s podobno sintakso, kot jo ima XML, in je namenjen izdelavi uporabniških vmesnikov [2]. Uporabniški vmesnik brskalnika Firefox je sestavljen iz skupine elementov XUL (kot so gumbi, okna in meniji), ki so definirani z enim ali več dokumenti XUL. Za dodajanje novih funkcionalnosti uporabniškega vmesnika brskalnika Firefox uporabimo t.i. prekrivno datoteko XUL (angl. *XUL overlay file*). Ta lahko vsebuje tudi skripte JavaScript, ki dodajo dinamično interakcijo novim elementom. Za razvoj vtičnika za brskalnik Firefox smo uporabili prav ta način. Več o prekrivnih datotekah bomo povedali v poglavju 4.3.1.

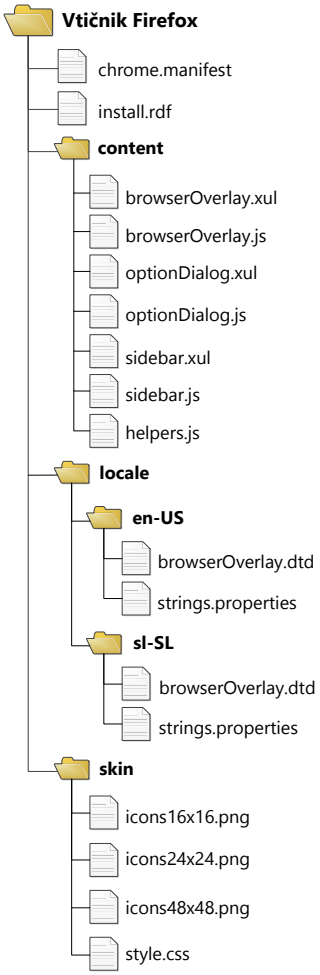
4.2 Zgradba in sestava vtičnika

Namenska mapa vtičnika za brskalnik Firefox mora vsebovati datoteki *chrome.manifest* in *instal.rdf* ter tri podmape, poimenovane *content*, *locale* in *skin* [24]. Da vtičnik lahko namestimo v brskalnik, mora biti celotna mapa zapakirana v datoteko XPI. XPI je arhivski format, ki je povsem identičen formatu ZIP in ga najlažje dobimo tako, da mapo zapakiramo v formatu ZIP, nato pa dobljeni datoteki spremenimo končnico iz .zip v .xpi.

Zgradba našega vtičnika za brskalnik Firefox je prikazana na sliki 4.1. Podrobnejši opis posameznih datotek sledi v nadaljevanju.

install.rdf

Install.rdf je datoteka v formatu RDF, ki vsebuje vse potrebne informacije o vtičniku za njegovo uspešno namestitev v brskalnik. Vsi elementi morajo biti znotraj korenskega elementa *Description*. Na primeru 4.1 je prikazana celotna vsebina datoteke *install.rdf* iz našega vtičnika. Vtičniku smo z elementom *id* najprej določili enolični identifikator (*firefoxCNP@mozilla.org*), ki je običajno oblikovan v obliki e-poštnega naslova, četudi izbran naslov ne obstaja. Preko elementa *version* in *name* smo določili številko različice vtičnika in njegov naziv. Z elementom *type* smo določili vrsto dodatka – 2 pomeni vtičnik, 4 temo za brskalnik, 8 jezikovni paket itd. Ime in opis vtičnika ter avtorja smo želeli tudi lokalizirati, zato smo jih



Slika 4.1: Struktura korenske mape vtičnika za brskalnik Firefox

postavili znotraj elementa *localized*. Poleg smo dodali še element *locale*, ki vtičniku pove, kateremu jeziku je lokalizacija namenjena. Lokalizacijo bomo podrobnejše predstavili v poglavju 4.4. V datoteki *install.rdf* je obvezen tudi poseben element, poimenovan *targetApplication*, s katerim določimo ciljno aplikacijo vtičnika [25]. S predpisanim nizom znakov za element *id* smo določili, da bo vtičnik namenjen brskalniku Firefox, z elementoma *minVersion* in *maxVersion* pa smo določili najnižjo in najvišjo verzijo brskalnika, ki ju vtičnik podpira.

Primer 4.1: Vsebina datoteke install.rdf

```
<?xml version="1.0"?>

<RDF xmlns="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:em="http://www.mozilla.org/2004/em-rdf#">

  <Description about="urn:mozilla:install-manifest">
        <em:id>firefoxCNP@mozilla.org</em:id>
        <em:version>1.0.0</em:version>
        <em:type>2</em:type>
        <em:name>Cookie Notification Preventer</em:name>
        <em:iconURL>chrome://firefoxCNP/skin/icon.png</em:iconURL>
        <em:homepageURL>https://addons.mozilla.org/en-US/firefox/addon/
            cookie-notification-preventer</em:homepageURL>
        <em:localized>
                <Description>
                        <em:locale>en-US</em:locale>
                        <em:name>Cookie Notification Preventer</em:name>
                        <em:description>Simple extension for preventing web
                            site's cookies notifications.</em:description>
                        <em:creator>Aleš Papler</em:creator>
                </Description>
        </em:localized>
        <em:localized>
                <Description>
                        <em:locale>sl-SL</em:locale>
                        <em:name>Preprečevalec obvestil piškotkov</em:name>
                        <em:description>Preprost vtičnik, ki prepreči
                            prikazovanje obvestil za piškotke na spletnih
                            straneh.</em:description>
                        <em:creator>Aleš Papler</em:creator>
                </Description>
        </em:localized>
        <em:targetApplication>
                <Description>
                        <em:id>{ec8030f7-c20a-464f-9b0e-13a3a9e97384}</em:id
                            >
                        <em:minVersion>4.0</em:minVersion>
                        <em:maxVersion>32.*</em:maxVersion>
                </Description>
        </em:targetApplication>
  </Description>
</RDF>
```

chrome.manifest

V datoteki *chrome.manifest* registriramo nove komponente vtičnika in definiramo strukturo. Znotraj datoteke povemo, kako se imenujejo obvezne mape (*content*, *locale* in *skin*) in določimo njihovo pot. Definiramo še, da gre za prekrivni način – naša datoteka *browserOverlay.xul* bo prekrila brskalnikovo datoteko *browser.xul*. Registriramo še datoteko CSS, da lahko uporabimo stilske predloge za elemente brskalnikove orodne vrstice (ikona vtičnika). Na primeru 4.2 je prikazana vsebina datoteke *chrome.manifest*, kjer smo ji določili relativne poti ključnih map vtičnika (*content*, *skin* in *locale*), glavne prekrivne datoteke (*browserOverlay.xul*) in datoteke CSS.

Primer 4.2: Vsebina datoteke chrome.manifest

```
content    firefoxCNP                  content/
skin       firefoxCNP  classic/1.0     skin/
locale     firefoxCNP  en-US           locale/en-US/
locale     firefoxCNP  sl-SL           locale/sl-SL/

overlay chrome://browser/content/browser.xul  chrome://firefoxCNP/content/
    browserOverlay.xul

style chrome://global/content/customizeToolbar.xul chrome://firefoxCNP/skin/
    toolbar.css
```

4.3 Uporabljene tehnike in pristopi

4.3.1 Prekrivanje

Prekrivanje (angl. *overlaying*) je način, ko razvijalec za 'podlago' novih komponent uporabi obstoječe programske komponente. Ta način se uporablja tudi pri tradicionalni izdelavi vtičnikov za brskalnik Firefox, ko se s tem pristopom doda dodatne komponente na obstoječi uporabniški vmesnik brskalnika. Videz novih komponent definiramo z datotekami XUL. S skriptami JavaScript pa komponentam dodamo dinamičnost in interaktivnost z uporabnikom. Prekrivna datoteka XUL mora imeti korenski element poimenovan *<overlay>*, nato pa znotraj tega elementa dodajamo nove komponente. Na primeru 4.3 je prikazano, kako smo

preko prekrivne datoteke XUL dodali dva nova elementa v meni desnega klika brskalnika.

Primer 4.3: Vsebina prekrivne datoteke browserOverlay.xul

```
<?xml version="1.0"?>

<?xml-stylesheet type="text/css" href="chrome://global/skin/"?>
<?xml-stylesheet type="text/css" href="chrome://firefoxCNP/skin/style.css"?>

<!DOCTYPE overlay SYSTEM "chrome://firefoxCNP/locale/browserOverlay.dtd">

<overlay id="CNP-overlay"
  xmlns="http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul">

        <script type="application/x-javascript"
                src="chrome://firefoxCNP/content/browserOverlay.js" />

        <stringbundleset id="firefoxCNPstringBundleSet">
                <stringbundle id="localStrings"
                  src="chrome://firefoxCNP/locale/strings.properties" />
        </stringbundleset>
        <!--Nova elementa v meniju desnega klika-->
        <menupopup id="contentAreaContextMenu">
                <menuitem label="&contextmenu.addEntry.labelC;"
                  id="contextMenuClick" class="menuitem-iconic"
                  oncommand="BrowserOverlay.saveToSQLite(gContextMenu.target
                       , 'click');" />
                <menuitem label="&contextmenu.addEntry.labelH;"
                  id="contextMenuHide" class="menuitem-iconic"
                  oncommand="BrowserOverlay.saveToSQLite(gContextMenu.target
                       , 'hide');" />
        </menupopup>
</overlay>
```

Nova elementa menija desnega klika smo določili z elementom XUL <*menuitem*> in ju nato postavili znotraj elementa <*menupopup*>. Določili smo jima še lokaliziran napis, enolični identifikator, stil in funkcijo, ki se kliče, ko je izbran eden izmed elementov.

4.3.2 Stranska vrstica in pogovorno okno z možnostmi

Znotraj našega vtičnika smo implementirali tudi stransko vrstico (angl. *sidebar*), kjer uporabnik lahko spreminja splošne nastavitve vtičnika in si ogleda ali uredi

shranjene zapise. Za primer lahko podamo dve obstoječi stranski vrstici, ki ju ima brskalnik Firefox že v osnovi. To sta stranska vrstica za pregled in urejanje zgodovine brskanja ter vrstica za zaznamke. Odločili smo se, da bomo na podoben način tudi mi izdelali stransko vrstico za potrebe našega vtičnika.

Za izdelavo stranske vrstice smo najprej potrebovali ločeno datoteko XUL (poimenovali smo jo *sidebar.xul*), ki smo jo v vtičnik vključili preko elementa <*menupopup*> znotraj glavne prekrivne datoteke *browserOverlay.xul*. Z atributom *label* smo določili napis, ki se pojavi v meniju *Pogled, Stranska vrstica*. Preko tega menija lahko odpremo oziroma zapremo stransko vrstico. Atribut *sidebarurl* pove brskalniku, kje se nahaja datoteka XUL s celotno vsebino stranske vrstice, atribut *oncommand* pa pove, katero stransko vrstico naj brskalnik odpre oziroma zapre.

Želeli smo tudi, da se stranska vrstica vtičnika odpre preko kombinacije tipk. Izbrali smo si kombinacijo *Alt + F5* in se s tem izognili morebitnemu konfliktu, saj je nobena druga komponenta v brskalniku še ne uporablja. Kombinacijo tipk smo nastavili v elementu <*key*>, nato pa smo ID tega elementa priredili atributu *key* znotraj elementa stranske vrstice. Vključitev stranske vrstice v prekrivni datoteki in registracija bližnjičnih tipk je prikazana na delu programske kode v primeru 4.4.

Primer 4.4: Vključitev stranske vrstice v glavno prekrivno datoteko

```
<menupopup id="viewSidebarMenu">
<menuitem id="CNPsidebar"
          label="&sidebar.title;"
          autoCheck="false"
          type="checkbox"
          group="sidebar"
          sidebarurl="chrome://firefoxCNP/content/sidebar.xul"
          sidebartitle="&sidebar.title;"
          key="sidebar-key"
          oncommand="toggleSidebar('CNPsidebar');" />
</menupopup>
<keyset>
          <key id="sidebar-key" modifiers="alt" keycode="VK_F5"
            oncommand="toggleSidebar('CNPsidebar');" />
</keyset>
```

V glavni prekrivni datoteki smo vključili stransko vrstico preko elementa XUL <*menuitem*>, ki je moral biti vsebovan znotraj elementa <*menupopup*>. Elementu *menuitem* smo določili še enolični identifikator, napis, relativni naslov, kjer je prekrivna datoteka za stransko vrstico, in funkcijo, ki se kliče, ko je stranska

vrstica izbrana.

Stranski vrstici smo dodali tudi dva zavihka. Prvi zavihek, ki smo ga poimenovali *Splošno*, vsebuje glavno stikalo vtičnika in gumb za izbris vseh uporabniških podatkov. Drugi zavihek smo poimenovali *Shranjeni podatki*, vsebuje pa tabelo s shranjenimi spletnimi elementi, ki jih je uporabnik dodal preko menija desnega klika. Vrstica v tabeli predstavlja en shranjen element.

Za potrebe urejanja in brisanja posameznih elementov smo izdelali pogovorno okno z možnostmi (angl. *option dialog*), katerega vsebina je določena z novo prekrivno datoteko XUL. Pogovorno okno se odpre, ko uporabnik izbere eno izmed vrstic tabele, v kateri se nahajajo shranjeni elementi. V oknu smo za potrebe urejanja potrebovali tudi vsebino izbrane vrstice. To smo rešili programsko, saj smo funkciji za odpiranje okna preko parametra prenesli želene podatke. Preko parametrov funkcije *window.openDialog* definiramo tudi, kje se prekrivna datoteka nahaja ter kakšen bo videz in položaj okna. Na primeru 4.5 je prikazano, kako programsko odpremo pogovorno okno in kako mu prenesemo podatke, ki jih želimo urediti.

Primer 4.5: Odpiranje pogovornega okna z možnostmi in prenos podatkov

```
// polje selectedRowValues ima format:
// ["rowId", "url", "elementId", "elementName", "elementClass", "action"]
var selectedRowValues;

// programsko odpremo pogovorno okno,
// prvi parameter je relativna pot prekrivne datoteke privzetega okna,
// z ostalimi parametri oknu nastavimo lastnosti
var optionDialog = window.openDialog(
        'chrome://firefoxCNP/content/optionDialog.xul',
                '', 'chrome,dialog,centerscreen,modal',selectedRowValues);

// ko želimo v skripti optionDialog.js dostopati do url-ja vrstice
// to storimo preko argumentov okna (objekt window)
var elementUrl = window.arguments[0][1];
```

4.3.3 Opazovalec–tema

Ob nekaterih dogodkih med delovanjem vtičnika smo želeli, da skripte med seboj 'komunicirajo'. Tak primer je, ko uporabnik preko menija desnega klika doda nov element, pri tem pa ima odprto stransko vrstico vtičnika. Opazili smo namreč, da

se tabela shranjenih elementov v stranski vrstici ne posodobi samodejno. Stransko vrstico je bilo potrebno ročno zapreti in ponovno odpreti, da se je tabela s podatki osvežila. To težavo smo rešili z uvedbo tako imenovanega opazovalca in teme (angl. *observer-topic*).

Osnovna ideja je bila, da skripta *browserOverlay.js*, ki zazna uporabnikovo dodajanje novega elementa, obvesti skripto *sidebar.js*, odgovorno za stransko vrstico, naj ponovno osveži tabelo s shranjenimi elementi. Na eni strani je bilo potrebno implementirati poslušalca na točno določeno temo, na drugi strani pa kreirati temo enake vrste in jo nato poslati vsem poslušalcem.

V skripti *browserOverlay.js* smo tik po shranjevanju novega elementa izdelali novo temo vrste *newEntryAdded-topic* in jo nato poslali vsem poslušalcem. Poleg bi lahko dodali še kakšen dodaten parameter, vendar to v našem primeru ni bilo potrebno. V skripti *sidebar.js* pa smo izdelali poslušalca na temo vrste *newEntryAdded-topic*, ki ob prejemu sporočila kliče funkcijo za osveževanje tabele v stranski vrstici. Primer 4.6 je izsek programske kode iz omenjenih skript, ki uporabljata koncept opazovalca in teme. Na strani opazovalca je skripta *sidebar.js*, na strani pošiljatelja teme pa skripta *browserOverlay.js*.

Primer 4.6: Uporaba koncepta opazovalec-tema

```
// izsek programske kode iz browserOverlay.js
var observerService = Components.classes["@mozilla.org/observer-service;1"].
    getService(Components.interfaces.nsIObserverService);
observerService.notifyObservers(null, "newEntryAdded-topic", null);

// izsek programske kode iz sidebar.js
var observerService = Components.classes["@mozilla.org/observer-service;1"].
    getService(Components.interfaces.nsIObserverService);
observerService.addObserver(observer, "newEntryAdded-topic", false);
var observer = {
  observe : function(aSubject, aTopic, aData) {
    if (aTopic == "newEntryAdded-topic") {
      refreshSidebarTable();
    }
  }
}
```

4.3.4 Hramba podatkov SQLite

Kot že omenjeno, smo se pri hrambi podatkov vtičnika odločili za uporabo SQLita. Branje, urejanje in brisanje zapisov v SQLiteu poteka preko klicev funkcij programskega vmesnika *mozIStorageConnection*. Klici teh funkcij so namenjeni le vtičnikom in komponentam brskalnika Firefox. Osnovni postopek za delo s hrambo podatkov SQLite znotraj vtičnika poteka po naslednjih korakih:

- v skripto uvozimo knjižnici *Services.jsm* in *FileUtils.jsm* za delo z s podatkovno bazo in z datotekami,
- pridobimo referenco na datoteko SQLite, ki se običajno nahaja v mapi profila brskalnika,
- odpremo povezavo s podatkovno bazo preko reference, dobljene v prejšnjem koraku,
- preko povezave ustvarimo poizvedbo, ki se bo izvedla nad hrambo podatkov,
- povežemo morebitne parametre (angl. *parameters binding*) s poizvedbo,
- izvršimo poizvedbo sinhrono (za preproste poizvedbe) oziroma asinhrono,
- preverimo morebitne napake, in
- ponastavimo poizvedbo in zapremo povezavo s podatkovno bazo.

Zaporedno izvajanje vseh teh korakov je prikazano na primeru 4.7, ki je izsek programske kode našega vtičnika. Preko funkcije *createStatement* vmesnika *mozIStorageConnection* smo naredili poizvedbo, nanjo povezali parameter in nato asinhrono izvedli izvrševanje poizvedbe. Rezultate poizvedbe smo nato brali znotraj zanke *for*.

Primer 4.7: Branje iz hrambe podatkov SQLite

```
Components.utils.import("resource://gre/modules/Services.jsm");
Components.utils.import("resource://gre/modules/FileUtils.jsm")
...
var file = FileUtils.getFile("ProfD", ["FirefoxCNP.sqlite"]);
var dbConn = Services.storage.openDatabase(file);
```

```
var statement = dbConn.createStatement("SELECT * FROM StoredData WHERE Url =
    :u");
statement.params.u = "www.fri.uni-lj.si";
statement.executeAsync({
        handleResult: function(aResultSet) {
                for (var row = aResultSet.getNextRow(); row; row =
                    aResultSet.getNextRow()) {
                        var siteUrl = row.getResultByName("Url");
                        ...
                }
        },
        handleError: function(aError) {
                console.log("Error occurred while executing query.");
        }
});
statement.reset();
dbConn.asyncClose();
```

Za izvrševanje poizvedb, ki ne vračajo rezultatov, pa uporabimo funkcijo *executeSimpleSQL*. Primer take poizvedbe iz našega vtičnika je prikazan na primeru 4.8, ko v podatkovni bazi kreiramo tabelo *Settings*, v primeru, da le-ta še ne obstaja.

Primer 4.8: Pisanje v hrambo podatkov SQLite

```
var file = FileUtils.getFile("ProfD", ["FirefoxCNP.sqlite"]);
var dbConn = Services.storage.openDatabase(file);

if (!dbConn.tableExists('Settings')){
  dbConn.executeSimpleSQL("CREATE TABLE Settings (Id INTEGER PRIMARY KEY
      AUTOINCREMENT, SettingName TEXT, Value TEXT)");
}
dbConn.close();
```

4.4 Lokalizacija

Uporabniški vmesnik vtičnika in njegova obvestila je tudi možno lokalizirati glede na jezik brskalnika (podobno kot smo to že storili pri vtičniku za brskalnik Chrome). V našem vtičniku smo lokalizacijo podprli za angleščino in slovenščino.

Znotraj mape vtičnika smo najprej ustvarili novo mapo, poimenovano *locale*, in znotraj nje še dve podmapi, *en-US* za angleščino in *sl-SL* za slovenščino. Nato je bilo treba znotraj teh dveh podmap ustvariti še dve novi datoteki – eno za lokalizacijo komponent v prekrivni datoteki XUL (poimenovano *browserOverlay.dtd*),

drugo pa za lokalizacijo obvestil znotraj skript JavaScript (poimenovano *strings.properties*).

Datoteka *browserOverlay.dtd* je v formatu DTD, njeno vsebino pa brskalnik uporabi za prevode komponent vtičnika, kot so to na primer pojavno okno in njegova vsebina, napisi v stranski vrstici in elementi menija desnega klika. Na primeru 4.9 je prikazan del vsebine datotek *browserOverlay.dtd* za angleški in za slovenski jezik.

Primer 4.9: Del vsebine datoteke browserOverlay.dtd

```
<!ENTITY sidebar.title                    "Cookie Notification Preventer">
<!ENTITY sidebar.tab.general.title        "General">
<!ENTITY sidebar.tab.storedData.title     "Stored data">
<!ENTITY optionDialog.title               "Options Dialog">
...
<!ENTITY sidebar.title                    "Preprečevalec obvestil piškotkov">
<!ENTITY sidebar.tab.general.title        "Splošno">
<!ENTITY sidebar.tab.storedData.title     "Shranjeni podatki">
<!ENTITY optionDialog.title               "Dialog opcij">
```

Na primeru 4.10 je prikazano, kako smo uporabili lokalizacijo za napise zavihkov v stranski vrstici. Pri tem smo morali najprej v datoteki XUL vključiti datoteko DTD, ki vsebuje prevode, za uporabo napisa pa je pred imenom napisa potrebno dodati le znak &.

Primer 4.10: Lokalizacija znotraj prekrivne datoteke

```
<!DOCTYPE overlay SYSTEM "chrome://firefoxCNP/locale/browserOverlay.dtd">
<page id="CNPSidebar"
  xmlns="http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul" >
        ...
        <tabbox id="CNPTabbox">
                <tabs>
                        <tab label="&sidebar.tab.general.title;" />
                        <tab label="&sidebar.tab.storedData.title;" />
                </tabs>
        </tabbox>
        ...
</page>
```

Preko atributa *em:localized* v datoteki *install.rdf* pa lahko lokaliziramo ime, opis in domačo stran vtičnika ter imena razvijalcev, prevajalcev in donatorjev. Za naš vtičnik smo naredili lokalizacijo za ime, opis in razvijalca vtičnika, kot je

prikazano že na primeru 4.1. Datoteka *install.rdf* pa mora biti v primeru uporabe šumnikov shranjena v formatu UTF-8.

4.5 Lokalno nameščanje vtičnika

Vtičnik smo za potrebe testiranja in razhroščevanja namestili v brskalnik iz našega datotečnega sistema. Datoteko XPI, ki vsebuje vse datoteke vtičnika, primemo in potegnemo v odprto okno brskalnika Firefox. Ko je namestitev končana, se brskalnik ponovno zažene. Drugi način lokalne namestitve je preko *Upravitelja dodatkov*, ki je dostopen preko menija *Orodja, Dodatki* ali preko kombinacije tipk *Ctrl* + *Shift* + *A*.

V Upravitelju dodatkov lahko začasno tudi onemogočimo vtičnik ali pa ga v celoti odstranimo iz brskalnika. Slabost lokalnega nameščanja vtičnika za brskalnik Firefox v primerjavi z lokalnim nameščanjem vtičnika za brskalnik Chrome je ravno v tem, da je treba za vsako programsko spremembo narediti novo datoteko XPI, jo prenesti v brskalnik in ga ponovno zagnati.

4.6 Razhroščevanje

Razhroščevanje vtičnika za brskalnik Firefox je v osnovi podobno kot razhroščevanje spletnih strani ali vtičnika za brskalnik Chrome. Na voljo je veliko brezplačnih orodij za razhroščevanje, v osnovi pa zadoščajo že razvijalska orodja, ki so del brskalnika Firefox. Z dodatkom DOM Inspector [12] pa lahko pregledamo tudi celoten DOM brskalnika in vtičnikov, kar se je pri našem razvoju izkazalo za zelo uporabno.

V primeru morebitne napake vtičnika se bo le-ta izpisala v brskalnikovi konzoli. Pri brskalniku Firefox nimamo štirih ločenih konzol kot pri brskalniku Chrome, temveč le eno enotno konzolo, kamor se izpisujejo vse morebitna sporočila in težave spletnih strani in vtičnikov.

4.7 Spletna trgovina Mozilla Add-ons

Mozilla Add-ons[2] je Mozillina uradna spletna trgovina z dodatki za Mozilline izdelke (brskalnik Firefox in SeaMonkey ter poštni odjemalec Thunderbird) [21]. Dodatki, kot so na primer vtičniki, razširitve, dodatne teme, slovarji ali jezikovni paketi, dodajo nove funkcionalnosti brskalniku oziroma aplikaciji in tako olajšajo delo uporabniku. V spletni trgovini so dodatki v osnovi brezplačni za namestitev in uporabo, vendar pri nekaterih obstaja tudi možnost, da uporabniki prostovoljno prispevajo določeno denarno vsoto, kot znak podpore ustvarjalcem.

Izkoristili smo možnost brezplačnega gostovanja in objave vtičnikov v Mozillini trgovini in to podrobneje opisali v poglavju 4.7.1.

4.7.1 Objava vtičnika v spletni trgovini

Pred objavo vtičnika v trgovini je bilo potrebno ustvariti uporabniški račun in nato preko spletnega obrazca[3] naložiti datoteko vtičnika XPI. Sledili so samodejni validacijski testi, ki so zajemali področja varnosti, lokalizacije, združljivosti in integracije vtičnika v brskalnik. Po uspešno izvedenih testih smo dodali kratek povzetek in podrobnejši opis vtičnika ter ga uvrstili v eno izmed kategorij vtičnikov. V naslednjem koraku smo dodali še ikono vtičnika, ki je prikazana v spletni trgovini, in nekaj zaslonskih posnetkov, ki prikazujejo delovanje vtičnika. Izbrali smo še eno izmed licenc, po katerih je licencirana izvorna koda, nato pa poslali celotno vsebino v pregled. Na voljo smo imeli dve vrsti pregleda, popolni pregled in predhodni pregled.

Pri popolnem pregledu Mozillini recenzenti pregledajo celotno programsko kodo, preverijo, če dodatek deluje v skladu s pričakovanji in če spoštuje varnostno politiko. Pregled lahko traja do deset dni in je bolj primeren za dodatke, ki so namenjeni širši množici. Predhodni pregled pa je namenjen eksperimentalnim dodatkom, saj recenzenti dodatka ne pregledajo povsem podrobno. Preverijo le, če izvorna koda dodatka ne krši pravil s stališča varnosti in veljavnega Mozillinega pravilnika. Rezultati pregleda so na voljo v treh dneh. Dodatek bo po predhodnem

[2] Dostopno na spletnem naslovu https://addons.mozilla.org.

[3] Dostopno na spletnem naslovu https://addons.mozilla.org/en-US/developers/addon/submit.

pregledu na voljo v galeriji spletne trgovine, le da bo imel opozorilo o opravljenem predhodnem pregledu in bo v galeriji uvrščen nižje kot dodatki, ki so prestali popolni pregled.

Mi smo se odločili za predhodni pregled, saj je vtičnik mišljen le kot demonstracija razvoja in primarno ni namenjen širšim množicam. Predhodni pregled je naš vtičnik uspešno prestal v manj kot enem dnevu po objavi osnutka. Končna verzija vtičnika je na voljo v spletni trgovini, kar prikazuje tudi slika 4.2. Do vtičnika pa je možen tudi direkten dostop preko spletnega naslova https://addons.mozilla.org/en-US/firefox/addon/cookie-notification-preventer.

Slika 4.2: Objava vtičnika v spletni trgovini Mozilla Add-ons

Poglavje 5

Prikaz delovanja in medsebojna primerjava razvojev vtičnikov

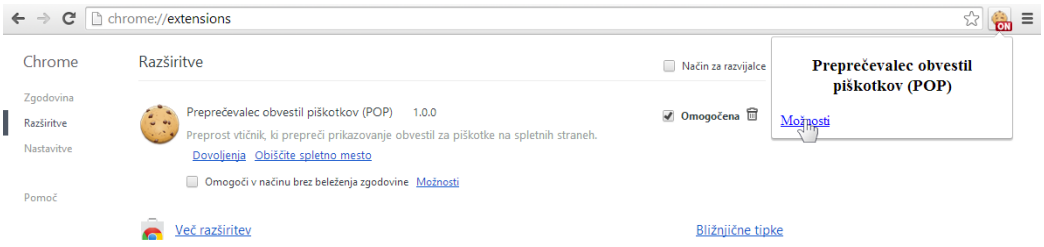
V tem poglavju smo opisali, kako prenesti in naložiti naš vtičnik iz spletne trgovine v brskalnik ter kako uporabljati vtičnik med brskanjem na spletu. V poglavju 5.3 pa smo napisali še nekaj besed o medsebojni primerjavi obeh vtičnikov in njunega razvoja.

5.1 Google Chrome

Za namestitev vtičnika za brskalnik Chrome obiščemo spletno trgovino in odpremo stran vtičnika (prikazana je na sliki 3.4). S klikom na gumb *+ Brezplačno* in po potrditvi se vtičnik samodejno namesti v brskalnik. Po namestitvi vtičnika nam ni potrebno ponovno zagnati brskalnika.

V orodni vrstici brskalnika se nato pojavi ikona vtičnika, kot je prikazano v desnem zgornjem kotu slike 5.1. Ob kliku na ikono se nam pokaže pojavno okno, v katerem se nahaja povezava na stran z možnostjo vtičnika. Do te strani pa lahko dostopamo tudi preko brskalnikovega menija *Orodja*, *Razširitve*, nato pa kliknemo na povezavo *Možnosti*, ki je pod opisom vtičnika.

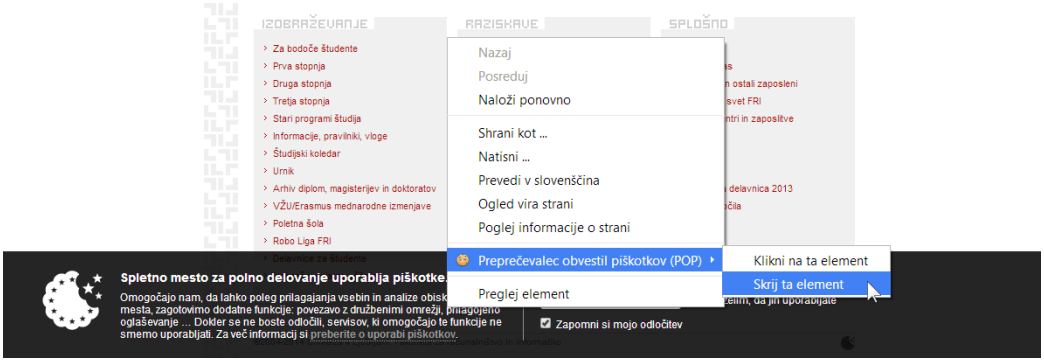
Vtičnik doda tudi dva nova elementa v meni desnega klika brskalnika. Prvi element se imenuje *Klikni na ta element*, drugi pa *Skrij ta element*. Če želimo, da vtičnik ob vsakem obisku ali ob osvežitvi neke spletne strani samodejno klikne na



Slika 5.1: Videz vtičnika, nameščenega iz spletne trgovine Chrome

element spletne strani (npr. na gumb, povezavo ali sliko), potem se z miško postavimo na ta element ter z desnim klikom izberemo možnost *Klikni na ta element*. Vtičnik bo nato dodal vnos z elementom v brskalnikovo hrambo podatkov in bo ob naslednji osvežitvi strani samodejno kliknil na ta element. Ob vsakem novem dodajanju elementa v hrambo podatkov se prikaže tudi obvestilo, ki se pojavi v desnem spodnjem kotu uporabnikovega zaslona.

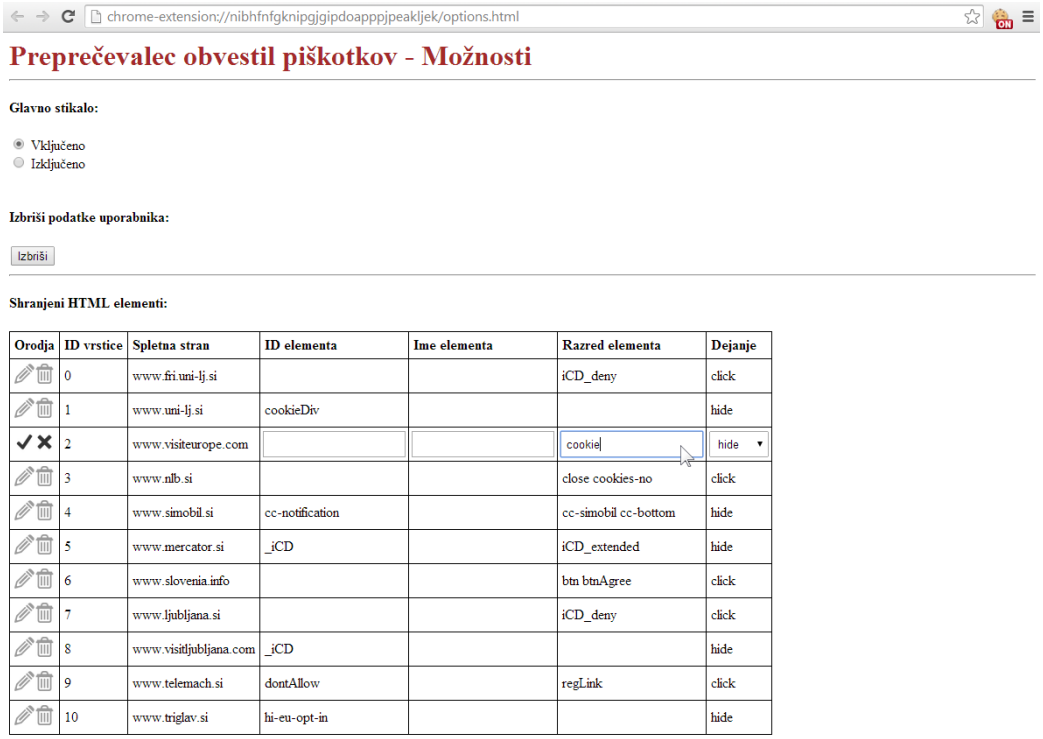
Podobno velja za drugo možnost – *Skrij ta element*. To izberemo takrat, ko želimo, da vtičnik ob vsakem obisku ali ob osvežitvi neke spletne strani samodejno skrije njen element (npr. pojavno okno, napis ali sliko). Na sliki 5.2 je prikazano, kako preko menija desnega klika dodamo nov element v hrambo podatkov.



Slika 5.2: Dodajanje novega elementa, vtičnik za brskalnik Chrome

Na strani možnosti vtičnika lahko pregledamo, uredimo in izbrišemo shranjene elemente. Elementu lahko uredimo njegov ID, ime in razred ter mu določimo eno izmed dveh dejanj (samodejni klik ali samodejno skrivanje), ki jih bo vtičnik izvršil

ob nalaganju spletne strani, kjer se element nahaja. Urejanje shranjenega elementa in videz strani z možnostmi sta prikazana na sliki 5.3.



Slika 5.3: Stran možnosti vtičnika za brskalnik Chrome

Na strani možnosti se nahaja tudi tako imenovano glavno stikalo, s katerim začasno zaustavimo samodejno izvrševanje dejanj vtičnika. Stanje stikala je prikazano z značko na ikoni vtičnika v orodni vrstici brskalnika. Uporabnik ima tudi možnost, da izbriše vse podatke, ki se nahajajo v hrambi podatkov. To stori preko gumba, ki se prav tako nahaja na strani možnosti.

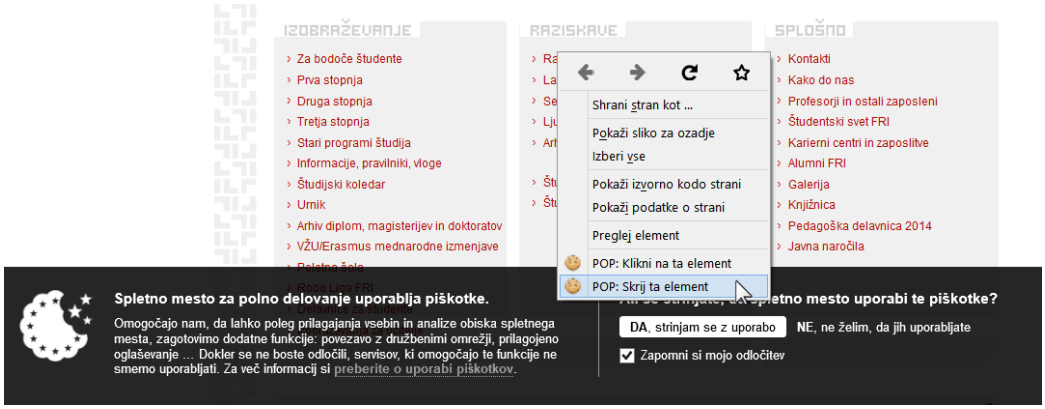
V primeru, da želimo v celoti odstraniti vtičnik iz brskalnika, to storimo preko ikone koša, ki se nahaja poleg opisa vtičnika na strani razširitev. Pri tem je treba opozoriti, da bodo vsi uporabniški podatki izbrisani.

5.2 Mozilla Firefox

Vtičnik za brskalnik Firefox namestimo preko Mozilline spletne trgovine z enim samim klikom. Na strani podrobnosti vtičnika le kliknemo na gumb *+ Add to Firefox* in tako bo vtičnik ob ponovnem zagonu brskalnika na voljo za uporabo.

Vtičnik bo ob prvem zagonu samodejno dodal svojo ikono v orodno vrstico brskalnika. Uporabnik lahko preko ikone odpira in zapira stransko vrstico. Do stranske vrstice pa lahko dostopamo tudi preko brskalnikovega menija *Pogled*, *Stranska vrstica* ali preko kombinacije tipk *Alt + F5*.

Na podoben način kot vtičnik za brskalnik Chrome tudi vtičnik za brskalnik Firefox doda dva nova elementa v meni desnega klika brskalnika. Funkciji teh elementov sta popolnoma enaki kot pri brskalniku Chrome. Edina razlika je le v tem, da brskalnik Firefox ne združi dveh ali več elementov menija v en korenski element, kot to stori brskalnik Chrome. Dodajanje novega elementa preko menija desnega klika je prikazano na sliki 5.4.

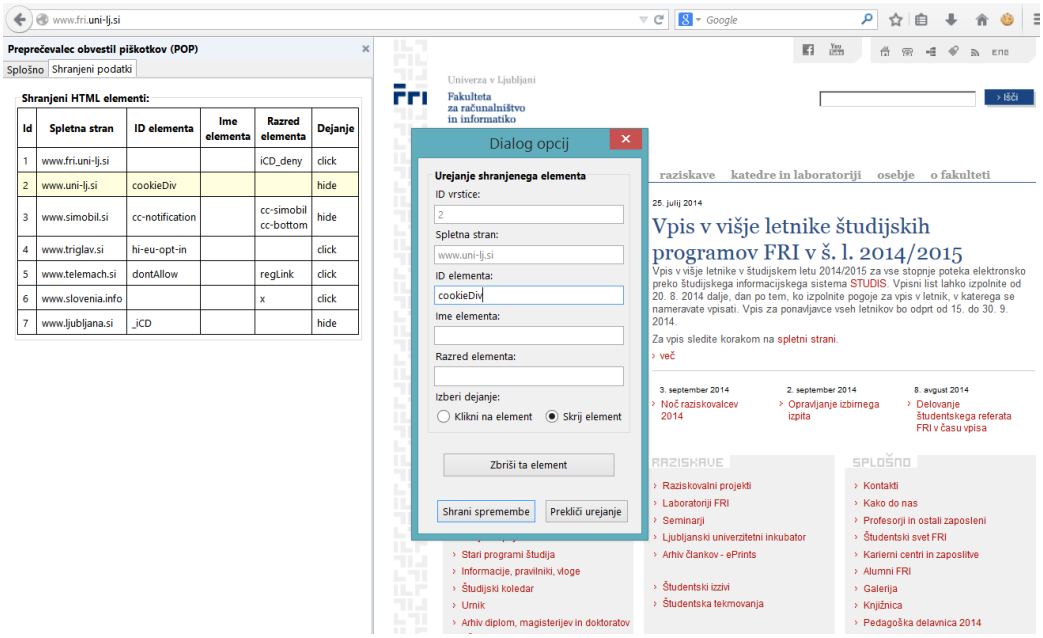


Slika 5.4: Dodajanje novega elementa, vtičnik za brskalnik Firefox

Preko glavnega stikala v stranski vrstici ima uporabnik možnost začasno zaustaviti samodejno izvrševanje dejanj vtičnika. Stil ikone vtičnika v orodni vrstici odraža stanje glavnega stikala. V primeru, da je glavno stikalo izključeno, je ikona obarvana sivo (angl. *grayed out*), v nasprotnem primeru pa je ikona v barvah. Pod zavihkom *Splošno* v stranski vrstici se nahaja tudi gumb za izbris vseh podatkov uporabnika. Zavihek *Shranjeni podatki* vsebuje tabelo, ki prikazuje shranjene

spletne elemente. S klikom na eno izmed vrstic tabele se vrstica obarva rumeno in odpre se pogovorno okno z možnostmi. Preko okna ima uporabnik možnost urediti zapis shranjenega elementa. Spremeni lahko ID, ime in razred spletnega elementa ter določi dejanje, ki ga vtičnik samodejno izvrši ob nalaganju spletne strani, kjer se element nahaja. Znotraj okna se nahaja tudi gumb, preko katerega lahko uporabnik izbriše posamezen zapis.

Na sliki 5.5 je prikazana stranska vrstica vtičnika in pogovorno okno z možnostmi, preko katerega lahko uporabnik ureja shranjene zapise.



Slika 5.5: Stranska vrstica in pogovorno okno z možnostmi vtičnika za brskalnik Firefox

5.3 Medsebojna primerjava razvoja vtičnikov

Razvoja vtičnikov za brskalnik Chrome in za brskalnik Firefox sta si v začetnih korakih podobna. Pri obeh smo naprej izdelali ogrodje, ki je bilo sestavljeno iz obveznih map in datotek. Pri tem so nam bili v pomoč primeri, ki se nahajajo na spletnih straneh za podporo razvijalcem. V ogrodje smo nato dodali želeno

vsebino vtičnika in določili njegovo interakcijo znotraj brskalnika.

V nadaljnjih korakih sta se razvoja pričela razlikovati, saj smo za brskalnik Chrome v večini primerov uporabili klice programskih vmesnikov, pri brskalniku Firefox pa je šlo za koncept prekrivanja in skripte v ozadju.

Prva težava, na katero smo naleteli pri razvoju za brskalnik Firefox, je bila, kako vključiti nekatere izmed skript JavaScript v eno ali več prekrivnih datotek XUL. Običajno vključevanje skript v prekrivno datoteko XUL preko elementa *<script>* ni delovalo pravilno. Težavo smo rešili tako, da smo izdelali ločeno skripto, ki je vsebovala le splošne funkcije, skupne vsem prekrivnim datotekam. Za komunikacijo med skriptami pa je bilo potrebno vpeljati koncept opazovalec-tema.

Druga težava se nam je pojavila pri lokalizaciji vtičnika za brskalnik Firefox. Potrebno je bilo namreč lokalizirati napise in obvestila v treh ločenih datotekah, kar je v začetku razvoja povzročilo zmedo. Poleg tega pa so morale biti datoteke zaradi šumnikov v slovenskem jeziku shranjene v formatu UTF-8, česar pa v uradni dokumentaciji nismo zasledili. Z vidika vzdrževanja je tak način težje obvladljiv, zato je način lokalizacije vtičnika za brskalnik Chrome v prednosti, saj so lokalizirani napisi shranjeni le v eni datoteki.

Prednost razvoja vtičnika za brskalnik Chrome v primerjavi z razvojem vtičnika za brskalnik Firefox je bila tudi v tem, da smo ob spremembi programske kode novo verzijo vtičnika naložili v brskalnik z enim samim klikom (gumb ≫*Vnovično nalaganje*≪ na strani razširitev). Za namestitev nove verzije vtičnika v brskalnik Firefox pa je bilo potrebno kreirati novo arhivsko datoteko XPI, jo prenesti v brskalnik in ga ponovno zagnati.

Kot prednost razvoja za brskalnik Firefox pa se je izkazala enotna konzola brskalnika, kamor so se izpisovala sporočila in morebitne težave spletnih strani in vtičnikov. Prav tako je preko enotne konzole lažje dostopati do virov vtičnika (npr. skript, slogovnih pol), kar omogoča lažje nastavljanje in opazovanje prekinitvenih točk pri procesu razhroščevanja.

Poglavje 6

Sklepne ugotovitve

V diplomski nalogi smo pokazali, kako razviti vtičnik za brskalnika Google Chrome in Mozilla Firefox. Razlog za razvoj vtičnika je bila nova evropska zakonodaja s področja elektronskih komunikacij, ki zahteva obvezen prikaz obvestila o uporabi piškotkov na spletnih straneh. Ravno ta obvestila pa so postala za obiskovalce spletnih strani moteča, zato smo si zadali cilj razviti vtičnik, ki bi prikaz omenjenih obvestil poskušal preprečiti oziroma skriti.

Najprej smo preučili veljavno evropsko in iz nje izpeljano slovensko zakonodajo. Nato smo se lotili razvoja vtičnika za brskalnik Google Chrome. Opisali smo osnove razvoja in uporabljene programske pristope, podprte z izseki programske kode celotne rešitve. Po končanem razvoju smo opisali tudi postopek objave vtičnika v spletni trgovini brskalnika. Podobno smo storili za brskalnik Mozilla Firefox. Po opisu osnov razvoja in uporabljenih tehnik z izseki programske kode celotne rešitve smo vtičnik objavili v brskalnikovi spletni trgovini in ta postopek tudi opisali. Sledil je prikaz delovanja obeh vtičnikov, na koncu pa še medsebojna primerjava obeh razvojev. Ugotovili smo, da sta si razvoja v osnovi podobna. Razlikujeta se le v načinu delovanja, lokalnega nameščanja, lokalizacije in procesa razhroščevanja.

Obe spletni trgovini brskalnikov razvijalcu posredujeta povratne informacije uporabnikov. Do sedaj še nismo prejeli nobene uporabniške ocene, je pa iz statističnih podatkov razvidno, da si naša dva vtičnika skupno ogleda do sto obiskovalcev dnevno. Od teh se jih nekaj deset odloči za namestitev.

Vtičnika imata zagotovo še prostor za izboljšave. Med prvimi izboljšavami bi

lahko razvili funkcionalnost, da bi vtičnik znal samodejno izvršiti dejanje v vseh odprtih zavihkih in oknih brskalnika. Trenutno lahko vtičnik samodejno izvrši dejanje le znotraj zavihka, ki ga ima uporabnik odprtega. Izboljšali bi lahko tudi sam videz uporabniškega vmesnika, saj mu do sedaj nismo namenili pretirane pozornosti, in uporabniku dali možnost personalizacije videza.

Literatura

[1] D. Crockford. JavaScript: The Good Parts. O'Reilly Media, 2008, poglavje 4.

[2] K. Feldt. Programming Firefox. O'Reilly Media, 2007, poglavja 2, 9, 11.

[3] J. Kreibich. Using SQLite. O'Reilly Media, 2010, poglavji 1, 2.

[4] W. Peng, J. Cisna. HTTP cookies – a promising technology, *Online Information Review*, št. 24, zv. 2, str. 150–153, 2000.

[5] Article 29 Working Party, Opinion 04/2012 on Cookie Consent Exemption (http://ec.europa.eu/justice/data-protection/article-29/documentation/opinion-recommendation/files/2012/wp194_en.pdf). Pridobljeno 23. 7. 2014.

[6] Direktiva 2009/136/ES evropskega parlamenta in sveta z dne 25. 11. 2009 (http://eur-lex.europa.eu/LexUriServ/LexUriServ.do?uri=OJ:L:2009:337:0011:0036:sl:PDF).

[7] Kdaj lahko uporabimo piškotke? Smenice informacijskega pooblaščenca (https://www.ip-rs.si/fileadmin/user_upload/Pdf/smernice/Smernice_o_uporabi_piskotkov.pdf). Pridobljeno 21. 7. 2014.

[8] Uradni list Evropske unije. Št. 201/37, 31. 7. 2002, str. 37 (http://publications.europa.eu/resource/cellar/cb5af945-9d9f-40e6-acee-bd0c5bb4ed0a.0020.02).

[9] Chrome Web Store, Wikipedia (http://en.wikipedia.org/wiki/Chrome_Web_Store). Pridobljeno 18. 8. 2014.

[10] Debugging, Wikipedia
(http://en.wikipedia.org/wiki/Debugging). Pridobljeno 17. 8. 2014.

[11] Developer Dashboard, Google Chrome
(https://chrome.google.com/webstore/developer/dashboard).

[12] DOM Inspector, Firefox Add-Ons
(https://addons.mozilla.org/en-US/firefox/addon/dom-inspector-6622).

[13] Firefox, Wikipedia
(http://en.wikipedia.org/wiki/Firefox). Pridobljeno 3. 9. 2014.

[14] Google Chrome, Wikipedia
(http://en.wikipedia.org/wiki/Google_Chrome). Pridobljeno 31. 7. 2014.

[15] HTTP cookie, Wikipedia
(http://en.wikipedia.org/wiki/HTTP_cookie). Pridobljeno 21. 7. 2014.

[16] Internationalizing Your App, Google Chrome
(https://developer.chrome.com/webstore/i18n?csw=1#localeTable).
Pridobljeno 17. 8. 2014.

[17] JavaScript APIs, Google Chrome
(https://developer.chrome.com/extensions/api_index).
Pridobljeno 10. 8. 2014.

[18] Komodo Edit, Wikipedia
(http://en.wikipedia.org/wiki/Komodo_Edit). Pridobljeno 5. 9. 2014.

[19] Manifest File Format, Google Chrome
(https://developer.chrome.com/extensions/manifest). Pridobljeno 5. 8. 2014.

[20] Message Passing, Google Chrome
(https://developer.chrome.com/extensions/messaging). Pridobljeno 13. 8. 2014.

[21] Mozilla Add-Ons, Wikipedia
(http://en.wikipedia.org/wiki/Mozilla_Add-ons). Pridobljeno 5. 9. 2014.

[22] StatCounter, Top 5 Desktop Browsers in Europe from Jan to Aug 2014 (http://gs.statcounter.com/#desktop-browser-eu-monthly-201401-201408).

[23] Storage, Google Chrome (https://developer.chrome.com/extensions/storage). Pridobljeno 12. 8. 2014.

[24] The Essentials of an Extension, Mozilla Developer Network (https://developer.mozilla.org/en-US/Add-ons/Overlay_Extensions/XUL_School/The_Essentials_of_an_Extension). Pridobljeno 28. 8. 2014.

[25] Valid Application Versions, Mozilla Add-Ons (https://addons.mozilla.org/en-US/firefox/pages/appversions). Pridobljeno 5. 9. 2014.

[26] Visual Studio, Wikipedia (http://en.wikipedia.org/wiki/Microsoft_Visual_Studio). Pridobljeno 8. 8. 2014.